Modellbasierte Einbettung von virtuellen Netzwerken in Rechenzentren

Tomaszek, Stefan

(2021)

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Modellbasierte Einbettung von virtuellen Netzwerken in Rechenzentren

VOM FACHBEREICH ELEKTROTECHNIK UND INFORMATIONSTECHNIK
DER TECHNISCHEN UNIVERSITÄT DARMSTADT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTOR-INGENIEURS (DR.-ING.)
GENEHMIGTE DISSERTATION

VON

STEFAN TOMASZEK

REFERENT: PROF. DR. RER. NAT. ANDY SCHÜRR
KORREFERENT: PROF. DR. RER. NAT. HABIL. UWE ASSMANN

TAG DER EINREICHUNG: 10.09.2020
TAG DER DISPUTATION: 01.12.2020

DARMSTADT 2020

Die Arbeit von Stefan Tomaszek wurde unterstützt durch den Sonderforschungsbereich (SFB) 1053 *Multi-Mechanismen-Adaptation für das künftige Internet (MAKI)* der *Deutschen Forschungsgemeinschaft (DFG)* (https://www.maki.tu-darmstadt.de).

Tomaszek, Stefan
Modellbasierte Einbettung von virtuellen Netzwerken in Rechenzentren
Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung der Dissertation auf TUprints: 2021
URN: urn:nbn:de:tuda-tuprints-173627
URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/17362
Tag der mündlichen Prüfung: 01.12.2020

ABSTRACT

The virtualization of network environments is a key technology for data center providers to operate hardware in a cost-effective and scalable manner and enabling these providers to make virtualized networks available to customers. The optimization problem for embedding virtual networks in data centers (VNE problem) is a fundamental problem for the efficient and cost-effective utilization of available hardware resources. For solving this problem we must consider the virtual network components as well as their topologies and further requirements. These requirements can vary from hardware-specific requirements (e.g. memory of virtual servers) to service-level agreements (e.g. availability of servers). To solve these dynamic and manifold VNE problems, both optimal and heuristic algorithms were presented, whose adaptation to different environments and requirements is a challenging task. In addition, the comparison of different algorithms is challenged by the absence of a standardized specification language and simulation environment.

Therefore, in this thesis we present a complete model-based development process to specify and solve VNE problems. For this purpose, we first develop a model-based specification language for the description of VNE problems based on a subset of UML (class diagram and OCL constraints). Based on this, we present two model-based approaches to solve VNE problems. In these approaches, model transformations and integer linear programming (ILP) are combined in such a way that the model transformation serves as search-space reducing technology and ILP finds a correct and optimal solution in this set of possible embeddings. Including the presented correct-by-construction methodology, a VNE algorithm can be derived from the model-based specification language, which guarantees correct and optimal solutions. A batch-based approach supports static application scenarios, in which only virtual networks are added, and an incremental approach supports dynamic scenarios, in which network components can also be changed or deleted. Thus, by these model-based approaches, the development effort can be shifted from (low-level) programming to (high-level) specification compared to handwritten implementations. In an experimental evaluation we can show that the computational efficiency of this model-based approach for static scenarios is comparable to a handwritten ILP-based implementation. For dynamic scenarios even a significant increase in efficiency could be demonstrated by the integration of incremental technologies, whereby the adaptation of algorithms for a static to a dynamic scenarios can be realized with small additional efforts.

KURZFASSUNG

Die Virtualisierung von Netzwerkumgebungen stellt für Rechenzentren eine Kerntechnologie zum kostengünstigen und skalierbaren Betrieb der Hardware dar und ermöglicht es, virtualisierte Netzwerke Kunden zur Verfügung zu stellen. Das Optimierungsproblem zur Einbettung von virtuellen Netzwerken in Rechenzentren (VNE-Problem) beschreibt hierbei eine grundlegende Problemstellung zur effizienten und kostengünstigen Nutzung der zur Verfügung stehenden Hardware. Bei dieser Problemstellung müssen neben den virtuellen Netzwerkkomponenten auch deren Topologien sowie weitere Anforderungen berücksichtigt werden. Diese Anforderungen können über hardwarespezifische Anforderungen (z.B. Arbeitsspeicher virtueller Server) bis hin zu Anforderungen aus Dienstleistungs-Güte-Vereinbarungen (z.B. Verfügbarkeit von Servern) reichen. Zur Lösung dieser dynamischen und vielfältigen VNE-Problemstellungen wurden sowohl optimale wie auch heuristische Algorithmen präsentiert, deren Anpassung an verschiedene Umgebungen und Anforderungen eine herausfordernde Aufgabe darstellt. Auch der Vergleich verschiedener Algorithmen wird durch das Fehlen einer einheitlichen Beschreibungssprache und Simulationsumgebung erschwert.

In dieser Arbeit präsentieren wir daher einen vollständigen Entwicklungsprozess, um VNE-Probleme einheitlich zu spezifizieren und zu lösen. Dafür entwickeln wir zunächst einen modellbasierten Spezifikationsansatz zur Beschreibung von VNE-Problemen basierend auf einer Teilmenge von UML (Klassendiagramm und OCL-Zusicherungen). Darauf aufbauend präsentieren wir zwei modellbasierte Ansätze zur Lösung von VNE-Problemen. Bei diesen Ansätzen werden Modelltransformationen und ganzzahlige lineare Programmierung (ILP) so miteinander kombiniert, dass die Modelltransformation als suchraumreduzierende Technologie dient und ILP in dieser Menge von möglichen Einbettungen eine korrekte und optimale Lösung findet. Unter Einbeziehung der präsentierten per-Konstruktion-korrekten Methodik kann nun aus dem modellbasierten Spezifikationsansatz ein VNE-Algorithmus abgeleitet werden, der korrekte und optimale Lösungen findet. Dabei unterstützt ein batchbasierter Ansatz statische Anwendungsszenarien, in welchem nur virtuelle Netzwerke hinzugefügt werden, und ein inkrementeller Ansatz dynamische Szenarien, in welchem Netzwerkkomponenten auch verändert oder gelöscht werden können. Durch diese modellbasierten Ansätze kann somit der Entwicklungsaufwand im Vergleich zu handgeschriebenen Lösungsstrategien von der (Low-Level) Programmierung hin zu einer (High-Level) Spezifikationstätigkeit verschoben werden. In einer experimentellen Evaluation zeigen wir, dass die Recheneffizienz des modellbasierten Ansatzes für statische Szenarien vergleichbar ist zu einer handgeschriebenen ILP-basierten Implementierung. Für dynamische Szenarien konnte sogar eine signifikante Steigerung der Effizienz durch die Integration von inkrementellen Technologien nachgewiesen werden, wobei die Anpassung von Algorithmen von einem statischen hin zu einem dynamischen Szenario mit einem geringen Mehraufwand realisiert werden kann.

INHALTSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

EMOF	Essential Meta Object Facility
EOCL	Essentail Object Constraint Language
ILP	Integer Linear Programming
MOF	Meta Object Facility
MT	Modelltransformation
OCL	Object Constraint Language
OMG	Object Management Group
TGG	Tripel-Graph-Grammatik
UML	Unified Modeling Language
VNE	Virtual Network Embedding
WB	Wissenschaftlicher Beitrag

1

EINLEITUNG

In der heutigen Zeit stellen Online-Services für mobile Anwendungen [103], vernetzte Spiele [13], Internethandel [81], Roboter und Automatisierungssysteme [52] und die Bereitstellung von Unternehmensanwendungen hohe Anforderungen an Service-Provider in Bezug auf Verfügbarkeit, Skalierbarkeit und Flexibilität. Die Verarbeitung und Speicherung der hohen Datenmengen lässt klassische Netzwerktopologien und deren Administration sowie die Verwaltung von Servern und Rechenzentren an ihre Grenzen stoßen. Eine Schlüsseltechnologie stellt hierbei die Bereitstellung von Services (Software, Plattform und Infrastruktur) in der Cloud dar [83]. Dadurch kann der Kunde selbstständig benötigte Speicherkapazitäten, Rechenleistungen oder Anwendungen konfigurieren und diese Ressourcen nach tatsächlicher Nutzung bezahlen. Um diesen Anforderungen an Verfügbarkeit, Skalierbarkeit, Flexibilität und kosteneffizienter Hardware- und Softwarenutzung gerecht zu werden, haben sich Rechenzentren als zentrale Infrastruktur etabliert [103]. Rechenzentren sind dabei mit einer Vielzahl von Servern, Switches und Speichersystemen ausgestattet, um die benötigte Rechenleistung, die Kapazitäten zur Verarbeitung und Speicherung der enormen Datenmengen und die notwendige Netzwerkinfrastruktur zur Verfügung zu stellen. Diese Bündelung von Ressourcen geht allerdings mit einer hohen Komplexität einher, die schon bei der Energieversorgung solcher Rechenzentren beginnt, über die Konfiguration und Verwaltung der Netzwerke und Datenflüsse geht und bei der dedizierten Nutzung einzelner Server, von verbundenen Speichersystemen und der Kostenabrechnung endet. Die Komplexität wird zusätzlich durch herstellerspezifische Eigenschaften erhöht, wodurch die Integration und Umsetzung von neuen Protokollen, Technologien oder prototypischen Entwicklungen eine enorme Herausforderung darstellt.

Die Virtualisierung von Rechenzentren stellt eine Kerntechnologie dar, um die umfangreichen und komplexen Umgebungen effizienter zu administrieren, da hierbei die physische Hardware durch virtuelle Komponenten abstrahiert, entkoppelt und standardisiert angesprochen werden kann. Durch diese Entkopplung der Anwendungen, Betriebssysteme und Konfigurationen können herstellerunabhängige Hardwarekomponenten, sowie automatisierte und vereinheitlichte Konzepte und Arbeitsabläufe zur Verwaltung, Änderung und Wartung der Netzwerke, Server und Speichersysteme genutzt werden. Somit wird die Migration von Anwendungen, z.B. die Verschiebung von Anwendungen auf andere Server, zur Laufzeit ermöglicht und sowohl die Hardware als auch die Infrastruktur können kosteneffizient betrieben werden. Auch der Energieverbrauch, ein hoher Kostenfaktor für Rechenzentren [90, 37], kann hierdurch gesenkt und die Auslastung der

Hardware verbessert werden. Dies setzt allerdings automatisierte Arbeitsabläufe innerhalb der virtuellen Umgebungen voraus, wodurch erst die hohen Anforderungen an Verfügbarkeit und Skalierbarkeit gewährleistet werden können. Diese automatisierten Arbeitsabläufe greifen hierbei in alle Ebenen im Betrieb von Rechenzentren ein. So wird neben der Energieversorgung auch die Netzwerk- und Switchkonfiguration administriert und sowohl Server als auch Anwendungen werden initialisiert, verwaltet und migriert. Dies stellt neue Herausforderungen an die benötigen Algorithmen und Technologien in den virtuellen Umgebungen dar [50].

Eine diese Herausforderungen betrifft die Einbettung virtueller Netzwerke, engl. Virtual Network Embedding (VNE), in Rechenzentren [28, 45]. Hierbei werden komplette (Kunden-)Netzwerke, im Weiteren als virtuelle Netzwerke bezeichnet, in das physische Netzwerk im Rechenzentrum platziert. Dabei bestehen die virtuellen sowie die physischen Netzwerke aus Servern mit Ressourcen (z.B. Prozessorkerne, Arbeitsspeicher oder Festplattenspeicher), Switches und Verbindungen (z.B. Ethernet oder Lichtwellenleiter). Zur Lösung dieser Problemstellung müssen nun alle virtuellen Komponenten unter Einhaltung von strukturellen, funktionalen und nicht-funktionalen Anforderungen im physischen Netzwerk bei gleichzeitiger Minimierung (Maximierung) einer Kostenfunktion platziert werden. Gängige Kostenfunktionen sind hierbei die Minimierung der Kommunikationskosten [118], des Energieverbrauchs [39] oder der verwendeten Hardware [123]. Somit stellt dies ein NP-hartes Optimierungsproblem [2] dar. Strukturelle Anforderungen beziehen sich hierbei auf die verwendete Netzwerktopologie, den vorhandenen oder angeforderten Hardwareressourcen (z.B. Prozessorkerne, Arbeitsspeicher oder Bandbreite) oder den hardwarespezifischen Funktionalitäten, die nur bestimmte Anwendungen auf dieser Hardware ermöglichen (z.B. Switches, Middleboxes oder Firewalls). Im Gegensatz hierzu beschreiben funktionale und nicht-funktionale Anforderungen Sicherheitsrichtlinien (z.B. Ausfallstrategien, Datenschutz oder Zugriffsberechtigungen), Service-Level-Agreements (z.B. Verfügbarkeit, Leistungsspektrum oder Reaktionszeiten) oder Geschäftsmodelle (z.B. Ertragsmodelle). In diesen dynamischen Umgebungen werden hohe Anforderungen an die Algorithmen gestellt, sodass hier ein Kompromiss zwischen der Laufzeit zur Lösung des Optimierungsproblems und dem Finden einer optimalen und garantiert korrekten Lösung bezüglich der strukturellen, funktionalen und nicht-funktionalen Anforderungen sowie dem Optimierungsziel eingegangen wird. Durch diese enorme Menge an Freiheitsgraden in den virtuellen und physischen Netzwerken, Anforderungen, Anwendungsfällen und Kostenfunktionen stellt die Entwicklung, Adaption, Simulation und der Vergleich von Einbettungsalgorithmen eine große Herausforderung dar.

Durch die wachsende Bedeutung von Cloud Computing und der Netzwerk- und Hardwarevirtualisierung wurde auch die Forschung zur Lösung des VNE-Problems intensiviert. Im Übersichtsartikel [28] werden die entwickelten Algorithmen in drei Typen kategorisiert: (i) Heuristische, (ii) metaheuristische und (iii) exakte Ansätze. Die heuristischen Ansätze (i) verfolgen hierbei das Ziel, den enormen Suchraum des VNE-Problems durch Ausnutzung und Reduktion der Eigenschaften in den Netzwerktopologien, Anforderungen, Anwendungsfällen und der

Kostenfunktion stark zu verkleinern. Dadurch kann die Laufzeit zur Lösung des Problems verringert und es können annähernd gute Einbettungen und Migrationsabläufe für diese speziellen Szenarien gefunden werden. Migrationen bezeichnen hierbei die Verschiebung von virtuellen Servern oder Switchen auf andere Substratserver oder -switche. Hierbei können diese Ansätze allerdings keine Garantien zur Einhaltung der Anforderungen oder Qualität der gefundenen Lösung (z.B. Optimalität) geben. Da diese Algorithmen in der Regel als Pseudocode vorliegen, stellt die manuelle Implementierung und Anpassung an verschiedene Endsysteme eine zusätzliche Fehlerquelle dar. Metaheuristische oder suchbasierte Ansätze (ii) nutzen einen problemunabhängigen Algorithmus, um eine annähernd gute Lösung zu finden, wobei auch hier keine Garantien zur Korrektheit der Lösungen gegeben werden können. Üblicherweise kommen hierfür genetische Algorithmen [71, 108] oder Verfahren aus der Partikelschwarmoptimierung [54, 121] zum Einsatz. Im Gegensatz dazu sind universell einsetzbare exakte (mathematische) Ansätze (iii) für (fast) alle Topologien, Anforderungen und Szenarien anwendbar, wobei eine etablierte Technologie in dieser Domäne die ganzzahlige lineare Programmierung, engl. Integer Linear Programming (ILP), darstellt. Die Vorteile dieser mathematischen Ansätze liegen in der garantierten Einhaltung aller Anforderungen (korrekte Lösung) und dem Finden einer optimalen Lösung. Allerdings benötigen diese Ansätze im Allgemeinen sehr lange zur Lösung des Problems, wodurch sie nur für kleine Rechenzentren geeignet sind [110]. Gerade bei dynamischen Umgebungen mit vielen, im Vergleich zum Gesamtsystem, kleinen (inkrementellen) Änderungen stellt die Entwicklung von Programmen zur Generierung und Anpassung von ILP-Formulierungen einen hohen Entwicklungsaufwand dar. Das beinhaltet auch die Sicherstellung, dass diese generierten ILP-Formulierungen weiterhin korrekte und optimale Lösungen gewährleisten können.

Die Entwicklung und vergleichbare Evaluation von Strategien zur Lösung des VNE-Problems in unterschiedlichen Szenarien stellt gerade in dynamischen Umgebungen eine große Herausforderung dar. Heuristiken können zwar sehr effizient dieses Problem lösen, lassen sich aber nur sehr schwer an neue Anwendungsfälle adaptieren, wo hingegen die mathematischen Verfahren zwar leicht anpassbar sind, aber sehr lange zum Lösen des Problems benötigen. Eine Kombination von suchraumreduzierenden und mathematischen Ansätzen stellt hierbei eine Möglichkeit dar, die Einflüsse der Suchraumreduktion von neuen Einbettungsstrategien in Hinblick auf Anpassbarkeit, Skalierbarkeit, Laufzeit, Korrektheit und Optimalität zu untersuchen und kann somit die Entwicklung, Umsetzung und Untersuchung von neuen Algorithmen erheblich erleichtern.

1.1 Wissenschaftliche Herausforderungen

In diesem Abschnitt stellen wir die wissenschaftlichen Herausforderungen vor, die sich bei der Entwicklung von Lösungsstrategien für dynamische VNE-Probleme in Rechenzentren ergeben und in dieser Arbeit behandelt werden.

Herausforderung 1: *Dynamische und vielfältige Anwendungsfälle*

Die Anwendungsfälle für VNE-Probleme in Rechenzentren sind sehr unterschiedlich gestaltet je nach eingesetzten Netzwerktopologien, Hardwarekomponenten, strukturellen und (nicht-)funktionalen Anforderungen oder Geschäftsmodellen. Zudem können sich alle Komponenten, Anforderungen oder Abrechnungsmodelle bzw. Zielfunktionen im laufenden Betrieb ändern, wodurch diese dynamischen und komplexen Systeme eine große Zahl an Freiheitsgraden, möglichen Einbettungen oder Migrationen besitzen. Dies kann beispielsweise ein Ausfall, die Wartung oder Änderung von physischer Hardware, die Neuerstellung, Anpassung oder das Löschen von virtuellen Komponenten, die Änderung von Topologien oder Anforderungen sein. Dabei reichen die Anforderungsänderungen von klar spezifizierten und (relativ) statischen Eigenschaften wie vorhandenen physischen Hardwareressourcen (z.B. verfügbare Arbeitsspeicher) über hardwarespezifischen Funktionseinschränkungen, die sich meist durch den Einsatz von Middleboxes (z.B. Firewalls), dedizierten Speichersystemen oder exklusiven Datenbankservern ergeben, über die Integration von statistischen Wahrscheinlichkeiten (z.B. mittlere Verfügbarkeiten oder Latenzzeiten in Service-Level-Agreements) bis hin zu systemübergreifenden Sicherheitsrichtlinien oder langfristigen Kostenbetrachtungen. Durch diese vielfältigen Anwendungsfälle ist die Anpassbarkeit von Algorithmen an die jeweiligen Systeme und Umgebungen von entscheidender Bedeutung und stellt durch die große Anzahl an Freiheitsgraden und Änderungsmöglichkeiten auch eine zentrale Herausforderung dar. Diese zentrale Herausforderung lässt sich in drei Fragestellungen unterteilen:

(a) Wie können dynamische VNE-Probleme mit deren jeweiligen Anforderungen in einer formalen Beschreibungssprache präzise spezifiziert werden?

(b) Wie können Algorithmen und die Qualität der daraus resultierenden Lösungen bewertet und quantifiziert werden?

(c) Wie können Algorithmen für diese Problemstellungen effizient umgesetzt, evaluiert und miteinander verglichen werden?

Die aus diesen Fragestellungen resultierenden Herausforderungen werden im Folgenden weiter erläutert.

Herausforderung 2: *Spezifikation von VNE-Problemen*

> *Wie können dynamische VNE-Probleme mit deren jeweiligen Anforderungen in einer formalen Beschreibungssprache präzise spezifiziert werden?*

Aktuell existiert weder für die Definition von VNE-Problemen für Rechenzentren noch für die Beschreibung von konkreten Anwendungsfällen mit deren Anforderungen und Kostenfunktionen eine etablierte und detaillierte Spezifikations- bzw. Beschreibungssprache. In der Literatur sind unterschiedliche Definitionen der VNE-Probleme zu finden, welche (mehr oder weniger starke) Auswirkungen auf die eingesetzten Algorithmen haben und ein Vergleich untereinander erschweren.

Dies betrifft beispielsweise die Ressourcen der Server, welche in einem Fall dezidiert mit Prozessorkernen, Arbeits- und Festplattenspeicher spezifiziert [119, 73] oder in einem anderen Fall als eine Menge von Slots für virtuelle Server [6, 112] abstrahiert werden. Aber auch die Datenpfade innerhalb des physischen Netzwerks [39, 109] oder die Kommunikation zwischen virtuellen Servern [66] werden nun in einigen Arbeiten zusätzlich zu den Hardwareressourcen mit berücksichtigt. Diese unterschiedlichen Definitionen des VNE-Problems werden hierbei noch in verschiedenen Beschreibungssprachen spezifiziert, welche von eher informellen Beschreibungen [6] bis hin zu einer mathematischen Formulierung mit linearen [5, 75] oder quadratischen [118] Ungleichungen variieren und dadurch schwer vergleichbar oder ineinander überführbar sind.

Herausforderung 3: *Qualität und Effizienz der VNE-Algorithmen*

> *Wie können Algorithmen und die Qualität der daraus resultierenden Lösungen bewertet und quantifiziert werden?*

Für die Entwicklung von Algorithmen stellt sich neben der Spezifikation der Problemstellung und der Umsetzung auch die Frage nach der Bewertung des entwickelten Algorithmus. Etablierte Bewertungskategorien für Algorithmen in dieser Domäne sind hierbei (i) Vollständigkeit, (ii) Korrektheit, (iii) Qualität und (iv) Effizienz. Neben dem Finden aller Lösungen für ein gegebenes Einbettungsproblem (Vollständigkeit) und der Einhaltung aller Anforderungen (Korrektheit) werden Metriken zur Quantifizierung der Qualität einer gefundenen Lösung (z.B. Abweichung vom Optimum) oder zur Bewertung der Effizienz (z.B. Laufzeit zum Finden dieser Lösung) für einen VNE-Algorithmus eingesetzt. Da viele Arbeiten Heuristiken zur Lösung der Problemstellung nutzen, sind Bewertungen zur Vollständigkeit, Korrektheit oder der Qualität nur sehr schwer realisierbar, da formale Eigenschaften in vielen Fällen nicht aufgezeigt werden. Auch die Quantifizierung der Qualität eines Algorithmus stellt ohne das Wissen der optimalen Lösungsmenge eine herausfordernde Aufgabe dar, welche durch die verschiedenen zum Einsatz kommenden Kostenfunktionen noch zusätzlich erschwert wird. Daher wird zur Bestimmung der Qualität eines VNE-Algorithmus oftmals die Anzahl von akzeptierten Netzwerkeinbettungen angewendet. Diese Metriken werden hierbei nur in seltenen Fällen in realistischen Testumgebungen evaluiert, wodurch meist Simulationsumgebungen zum Einsatz kommen, welche in der Regel für bestimmte Veröffentlichungen zugeschnitten sind.

Da in der Entwicklung von neuen Algorithmen oftmals ein Kompromiss zwischen der Effizienz und Qualität der gefundenen Lösungen unter Berücksichtigung der Korrektheit gesucht wird, spielt die Definition und Quantifizierung dieser Bewertungen in ausgiebigen Simulationen eine wichtige Rolle.

Herausforderung 4: *Rapid Prototyping von VNE-Algorithmen*

> *Wie können Algorithmen für diese Problemstellungen effizient umgesetzt, evaluiert und miteinander verglichen werden?*

Die Umsetzung der meisten VNE-Algorithmen wird manuell anhand einer gegebenen Problembeschreibung oder eines entwickelten Pseudocodes durchgeführt. Dabei werden ausführbare Programme für bestimmte Evaluationen, Umgebungen oder Veröffentlichungen erstellt, die nur schwer vergleichbar und anpassbar sind. Eine Integration in andere Simulationsumgebungen ist somit kaum möglich und auch die Sicherstellung der Korrektheit oder Vollständigkeit ist nur schwer realisierbar. Aber auch die Nutzung von etablierten Technologien wie ILP stellt hohe Anforderungen an die Umsetzung von VNE-Problemen, da die ILP-Formulierungen während der Laufzeit des Algorithmus für die konkrete Netzwerktopologie mit den jeweiligen Anforderungen und der Kostenfunktion generiert und eventuell angepasst werden müssen. Dies setzt ein hohes Maß an Domänenwissen aus dem Bereich ILP und VNE voraus und die Anpassung der ILP-Formulierungen stellt sich gerade in dynamischen Umgebungen als sehr fehleranfällig heraus.

Die Erstellung eines ausführbaren Programms für VNE-Algorithmen und vergleichbare Evaluationen ist deshalb durch die vielen manuellen Schritte sehr fehleranfällig, schwer anpassbar und vergleichbar.

1.2 Zielsetzung und Beitrag

Die übergeordnete Zielsetzung dieser Arbeit ist die Konzeption einer Methodik zur Spezifikation, Entwicklung und Umsetzung von dynamischen VNE-Algorithmen für Rechenzentren mithilfe von modellbasierten Konzepten und ILP-Technologien. Dazu werden die wissenschaftlichen Beiträge (WB) in die folgenden beiden Kategorien unterteilt:

1. Spezifikationen und Lösungsstrategien für VNE-Probleme
2. Rapid Prototyping von VNE-Algorithmen

Zur Veranschaulichung der einzelnen Beiträge, deren Abhängigkeiten und der Struktur dieser Arbeit dient Abbildung 1.1. Zusätzlich wird in dieser Abbildung der gesamte Arbeitsablauf und die Funktionsweise des Kernbeitrags dieser Arbeit abstrakt dargestellt. Im Folgenden werden nun die einzelnen Kategorien und Beiträge näher erläutert.

Spezifikationen und Lösungsstrategien für VNE-Probleme

Zur Entwicklung von Konzepten und Strategien zur Lösung von VNE-Problemen müssen diese als erstes klar spezifiziert und in geeigneter Weise beschrieben werden. Ausgehend von dieser Problemspezifikation kann dann die eigentliche Lösungsstrategie entwickelt und konzeptionell entworfen werden. Diese beiden Schritte stellen auch die beiden wissenschaftlichen Beiträge in dieser Kategorie dar.

WB 1: Modellbasierte Spezifikation zur Beschreibung von VNE-Problemen.

Zur Spezifikation von VNE-Problemen für Rechenzentren werden in dieser Arbeit universell einsetzbare Modellierungssprachen mit einer mathematisch formulier-

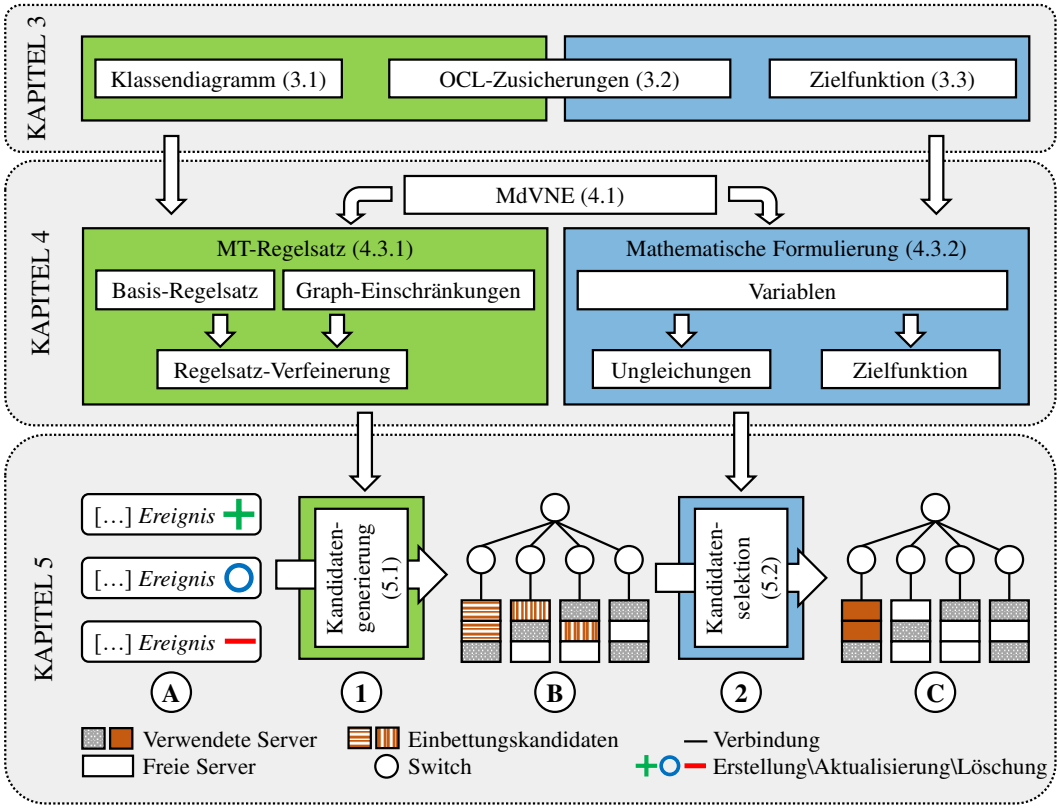


Abbildung 1.1: Wissenschaftliche Beiträge und Struktur dieser Arbeit

ten Zielfunktion verwendet. Dadurch können Anwendungsfälle mit verschiedene Anforderungen und Geschäftsmodellen spezifiziert werden. Zur grundlegenden Modellierung der strukturellen Eigenschaften wie Topologien, Server-, Switch- oder Verbindungskonfigurationen oder von möglichen Einbettungen werden Klassendiagramme aus der Unified Modeling Language (UML) [12] eingesetzt. Diese Klassendiagramme können durch Zusicherungen aus der Object Constraint Language (OCL) [38] mit weiteren (nicht-)funktionalen Anforderungen angereichert werden. Zur Abbildung des Geschäftsmodells wird eine mathematisch formulierte Zielfunktion verwendet, welche sich an ILP-Zielfunktionen orientiert.

WB 2: Modellbasierter Ansatz zur Lösung von VNE-Problemen.

In dieser Arbeit wird ein modellbasierter Ansatz zur Entwicklung von Algorithmen für dynamische und statische VNE-Probleme vorgestellt. Dieser Ansatz kombiniert Technologien zur Suchraumreduktion und ganzzahliger Optimierung, um sowohl heuristische als auch optimale und korrekte Algorithmen zu entwickeln und umzusetzen. Zur Reduzierung des Suchraums werden Modelltransformationen (MTs) verwendet, welche die virtuellen Netzwerke auf das Substratnetzwerk abbilden, wobei ein Regelsatz diese Transformationen spezifiziert. Diese Kombination von MT- und ILP-Technologien, integriert in ein einheitliches Konzept, kann sowohl garantiert alle Anforderungen einhalten (korrekte Lösungen) als auch optimale Lösungen in Bezug auf die Zielfunktion finden.

Rapid Prototyping von VNE-Algorithmen

In dieser Kategorie wird auf die systematische Konstruktion und die Generierung von ausführbarem Code für VNE-Algorithmen eingegangen. Dazu wird eine systematische Konstruktionsmethodik zur Erstellung von MT-Regeln und mathematischen Formulierungen (ILP) vorgestellt. Darauf aufbauend wird ausführbarer Code generiert und zusätzlich werden inkrementelle Technologien zur besseren Unterstützung von dynamischen Szenarien integriert. Dadurch können die, im Vergleich zum Gesamtsystem, kleinen Änderungen im virtuellen oder Substratnetzwerk oder deren Anforderungen effizient behandelt werden. Diese Schritte spiegeln sich auch in den drei wissenschaftlichen Beiträgen wider.

WB 3: Systematische Konstruktionsmethodik zur Erstellung eines Regelsatzes für MT und zusätzlichen mathematischen Formulierungen.

Ausgehend von der eingeführten modellbasierten Problembeschreibung wird in dieser Arbeit eine Konstruktionsmethodik vorgestellt, um einen MT-Regelsatz mit zusätzlichen mathematischen Formulierungen zu erstellen. Hierbei wird durch die verwendete Entwicklungsmethodik die Korrektheit (Einhaltung aller Anforderungen) und die Optimalität per Konstruktion sichergestellt. Dies wird anhand eines statischen VNE-Problems detailliert verdeutlicht.

WB 4: Integration von inkrementellen MT- und ILP-Technologien für dynamische VNE Anwendungsfälle.

Die Unterstützung von dynamischen Anwendungsfällen wird in dieser Arbeit durch die Integration von inkrementellen MT-Techniken und die Aktualisierung der ILP-Formulierung anhand der (meist kleinen) Modelländerungen weiter verbessert. Somit ist es möglich, dass nur auf die Änderungen in den konkreten Systemzuständen reagiert werden muss und mögliche im Vorfeld berechnete Lösungen und Datenstrukturen wiederverwendet werden können.

WB 5: Generierung von ausführbarem Code für modellbasierte VNE-Algorithmen.

Aus den durch die Konstruktionsmethodik erstellten MT-Regeln und ILP-Formulierungen wird ausführbarer Code generiert und in Simulationen evaluiert.

1.3 Aufbau der Arbeit

In diesem Abschnitt wird der Aufbau der Arbeit vorgestellt und es werden die wissenschaftlichen Beiträge den einzelnen Kapiteln zugeordnet. In Abb. 1.1 werden hierbei die einzelnen Kapitel und deren Themenbereiche nochmal im Gesamtzusammenhang verdeutlicht.

In Kapitel 2 stellen wir den grundlegenden Aufbau von physischen und virtuellen Netzwerken, deren Komponenten, Anforderungen und Einschränkungen vor, die für das Verständnis der Arbeit erforderlich sind, und erläutern Grundlagen der Virtualisierung von Netzwerken (Abschnitt 2.1). Danach wird auf das

Problem zur Einbettung von virtuellen Netzwerken in Rechenzentren eingegangen (Abschnitt 2.2). Dabei wird die für diese Arbeit zugrundeliegende Problemstellung als Graphrepräsentation und ILP-Formulierung vorgestellt und in der Literatur vorhandene Lösungsstrategien präsentiert. Diese Grundlagen werden anhand eines durchgängigen Beispiels weiter veranschaulicht.

Danach führen wir in Kapitel 3 die entwickelte modellbasierte VNE-Problemdefinition ein und erläutern diese anhand eines durchgängigen Beispiels. Dabei gehen wir auf die drei Komponenten der modellbasierten VNE-Problemdefinition ein: Klassendiagramm (Abschnitt 3.2), OCL-Zusicherungen (Abschnitt 3.3) und Optimierungsziel (Abschnitt 3.4). Dieses Kapitel beschreibt somit den wissenschaftlichen Beitrag **WB 1**.

In Kapitel 4 stellen wir das neuartige Konzept für den modellbasierten Ansatz (Abschnitt 4.1) und Grundlagen zu Tripel-Graph-Grammatiken (Abschnitt 4.2) vor. Danach präsentieren wir die systematische Konstruktionsmethodik in Abschnitt 4.3, wobei wir zuerst auf die Erstellung des MT-Regelsatzes eingehen (Abschnitt 4.3.1), danach die Erstellung einer mathematischen Formulierung (Abschnitt 4.3.2) vorstellen und zum Schluss die Korrektheit und Optimalität dieses Ansatzes beweisen (Abschnitt 4.3.3). Somit werden in diesem Kapitel die beiden wissenschaftlichen Beiträge **WB 2** und **WB 3** vorgestellt.

In Kapitel 5 setzen wir den entstandenen MT-Regelsatz und die zusätzlichen mathematischen Formeln in Form einer ILP-Formulierung in ausführbaren Quellcode um. Dabei wird zuerst die grundlegende Architektur zur Umsetzung des VNE-Algorithmus anhand der modellbasierten Problemdefinition vorgestellt und danach auf die Generierung der Kandidaten (Abschnitt 5.2) und die Selektion der Kandidaten (Abschnitt 5.3) näher eingegangen. Somit wird in diesem Kapitel **WB 4** und **WB 5** präsentiert.

In Kapitel 6 führen wir eine experimentelle Evaluation des entwickelten modellbasierten Konzeptes durch. Dazu stellen wir in Abschnitt 6.1 den Versuchsaufbau vor und evaluieren danach ein statisches (Abschnitt 6.2) und ein dynamisches Anwendungsszenario (Abschnitt 6.3). Zum Schluss vergleichen wir die beiden modellbasierten Ansätze zur Lösung des statischen und des dynamischen Anwendungsszenarios miteinander (Abschnitt 6.4).

Danach diskutieren wir in Kapitel 7 verwandte Arbeiten und schließen in Kapitel 8 diese Arbeit mit einer Zusammenfassung und einem Ausblick auf vielversprechende zukünftige Forschungsfelder ab.

2

GRUNDLAGEN

In diesem Kapitel stellen wir Technologien und grundlegende Konzepte vor, die für das Verständnis der vorliegenden Arbeit und den vorgestellten Herausforderungen notwendig sind. Dabei gehen wir in Abschnitt 2.1 auf den grundlegenden Aufbau von Netzwerken und deren Komponenten sowie auf die Virtualisierung von Netzwerken und Rechenzentren ein. Danach stellen wir in Abschnitt 2.2 die Problemstellung zur Einbettung von virtuellen Netzwerken in Rechenzentren als Graphrepräsentation und als mathematische Formulierung vor. Hierbei wird auch ein motivierendes Anwendungsszenario präsentiert, welches als durchgängiges Beispiel in dieser Arbeit dient. Danach geben wir einen Überblick über schon bestehende Methoden, Techniken und Algorithmen zur Lösung dieser Problemstellung.

2.1 Netzwerke und Virtualisierung

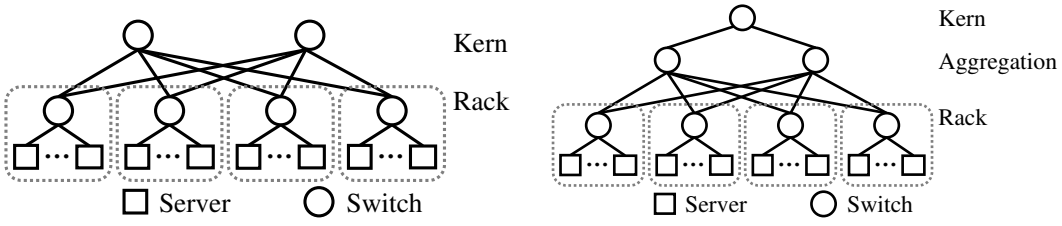
In diesem Abschnitt beschreiben wir sowohl den Aufbau als auch die Virtualisierung von Netzwerken und Rechenzentren. Dabei stellen wir zuerst in Abschnitt 2.1.1 die Netzwerke, deren Komponenten und Eigenschaften vor und geben einen Überblick über etablierte Topologien. Danach führen wir in Abschnitt 2.1.2 in die Virtualisierung von Netzwerken und Rechenzentren ein und zeigen Methoden zur Virtualisierung von Netzwerkkomponenten auf und beschreiben deren Eigenschaften.

2.1.1 *Aufbau von Netzwerken*

Netzwerke, auch Computer- oder Rechnernetzwerke genannt, bilden zentrale Einheiten zur Verarbeitung, Speicherung und Kommunikation von Daten [95]. Sie bestehen primär aus drei Komponenten: den *Servern* zur Verarbeitung und Speicherung der Daten, den *Switches*, zur Kontrolle der Datenflüsse innerhalb des Netzwerks und den *Verbindungen* zwischen diesen beiden Komponenten. Dabei werden Server üblicherweise durch die drei Kernbestandteile *Prozessor*, *Arbeitsspeicher* und *Festplatte* charakterisiert, die jeder Server in verschiedener Ausprägung besitzt. Natürlich können zusätzliche Bestandteile wie Grafikkarten, Netzwerkkarten oder Verschlüsselungschips in diesen Servern verbaut sein, um spezifische Funktionalitäten und Anwendungen realisieren zu können. Aber auch die Ausprägung des Prozessors (z.B. Anzahl der Prozessorkerne oder Taktung der Prozessoren), des Arbeitsspeichers (z.B. Größe, Taktung oder Zugriffszeiten) oder

der Festplatte (z.B. Größe oder Zugriffszeiten) ist für bestimmte Anwendungsfälle relevant. Zur Kommunikation nach außen sind Server mit Netzwerkkarten ausgestattet, die Verbindungen zu anderen Servern, Endgeräten oder Switches ermöglichen. Hierbei können sowohl kabelgebundene (z.B. Ethernet oder Lichtwellenleiter) als auch drahtlose Verbindungen (z.B. WLAN oder Freiraumoptik) zum Einsatz kommen, welche sich in der Bandbreite, der Verzögerungszeit, Verbindungsstabilität, Erweiterbarkeit oder den Installations- und Unterhaltskosten unterscheiden. So besitzen kabelgebundene Verbindungen im Allgemeinen eine höhere Bandbreite und sind stabiler im Betrieb, allerdings auch kostenintensiver in der Installation und schwerer zu erweitern als beispielsweise WLAN-Verbindungen. In klassischen Servernetzwerken kommen kabelgebundene Verbindungen zum Einsatz, die über Switches zusammengeschaltet werden. Diese Switches kontrollieren den Datenstrom und damit die Netzwerkpakete, die zur Kommunikation zwischen den Servern und den Endgeräten ausgetauscht werden. Grundlegende Eigenschaften dieser Switches stellen die Anzahl der Netzwerkanschlüsse und deren Bandbreite dar, da möglichst viele Daten in geringer Zeit verarbeitet und weitergeleitet werden sollen.

Große Netzwerke sind vor allem in Rechenzentren zu finden, die Rechenleistung, Speicherplatz und Ressourcen für Dienstleistungen (z.B. Cloud-Anwendungen) mit bis zu mehreren zehntausend Servern zur Verfügung stellen [14]. Diese Server werden dabei klassischerweise in Racks, Serverschränke mit einem gemeinsamen Switch, organisiert. Dies ermöglicht eine klar strukturierte Verkabelung aller Server innerhalb des Racks mit dem Rackswitch. Der Rackswitch ist wiederum mit anderen Switches verbunden, sodass baumartige Topologien mit den Servern als Blätter und den Switches als innere Knoten entstehen. Diese Topologien werden auch als *Tier-Netzwerke* bezeichnet, wobei ein Rack als 1-Tier-Netzwerk mit einer sternförmigen Verdrahtung, mit dem Rackswitch als Mittelpunkt, angesehen werden kann. In Abb. 2.1a ist ein 2-Tier-Netzwerk abgebildet, welches beispielsweise in kleineren Rechenzentren zum Einsatz kommt. Hierbei

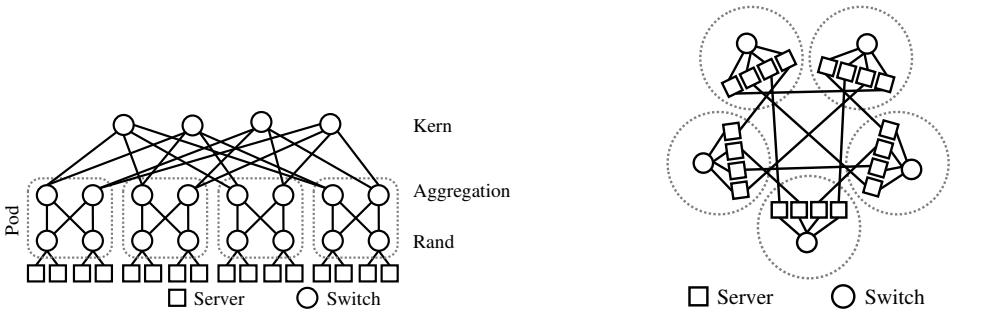


(a) 2-Tier-Netzwerktopologie **(b)** 3-Tier-Netzwerktopologie

Abbildung 2.1: 2- und 3-Tier-Netzwerktopologien

sind die Server als Quadrate, die Switches als Kreise und die kabelgebundenen bidirektionalen Verbindungen als Verbindungslinien zwischen den Servern und den Switches dargestellt. Beim 2-Tier-Netzwerk sind alle Rackswitches mit sogenannten Kernswitches verbunden, sodass die rackübergreifende Kommunikation von Servern oder der Zugriff zum Internet über diese Kernswitches geleitet wird. Da diese Netzwerke nicht für eine große Anzahl an Servern mit den benötigten Bandbreiten und Redundanzen skalieren, kommen in traditionellen Rechenzentren 3-Tier-Netzwerke zum Einsatz. Abbildung 2.1b präsentiert eine

solche Topologie, wobei die Anzahl der Switches in der Kern- und Aggregationsebenen je nach Rechenzentrum unterschiedlich ausfallen können. Auch hier sind die Rackswitches mit der nächst höheren Ebenen von Switches, sogenannten Aggregationsswitches, verbunden, die wiederum mit Kernswitches kommunizieren. Da diese Topologien einige Nachteile in Bezug auf Flexibilität, Erweiterbarkeit, Bandbreitenbeschränkungen, Komplexität der Verkabelung und der entstehenden Kosten aufweisen, wurde in den letzten 10 Jahre eine Vielzahl von neuen Topologien für Rechenzentren entwickelt. Eine bekannte Topologie, die aus der klassischen 3-Tier-Netzwerktopologie abgeleitet wurde, ist der Fat-Tree [1]. Diese Topologie, dargestellt in Abb. 2.2a, besitzt keine klassische Unterteilung in Racks mehr, sondern gliedert sich in Pods mit mehreren Switches (Rand und Aggregation). Spezifiziert wird sie über den Parameter k, der die Anzahl der Pods,



(a) Fat-Tree-Netzwerktopologie [1] **(b)** DCell-Netzwerktopologie [40]

Abbildung 2.2: Switch- und serverzentrierte Topologien

der Server und der Switches festlegt, wodurch verlässliche Aussagen (z.B. über die zur Verfügung stehende Bandbreite) getroffen werden können. Im Laufe der Zeit wurden noch viele andere switchzentrierte Netzwerktopologien entwickelt, in welchen die Switches für die Kontrolle der Datenströme eingesetzt werden. Einige Beispiele stellen hierbei VL2 [36], Diamond [94] oder F10 [64] dar. Neben den switchzentrierten Netzwerken wurden auch serverzentrierte Topologien entwickelt, in welchen nicht die Switches für die Verteilung der Datenströme zuständig sind, sondern die Server. Ein Beispiel hierfür stellt die DCell-Topologie [40] aus Abb. 2.2b dar. Diese Topologie ist in Zellen organisiert, wobei die Switches nur für die Datenströme innerhalb einer Zelle verantwortlich sind. Diese Zellen können rekursiv in größere Zellen strukturiert werden. Für eine ausführliche Übersicht und Kategorisierung von Rechenzentren und deren Topologien sei auf [14] verwiesen.

2.1.2 *Virtualisierung von Netzwerken und Rechenzentren*

Die Virtualisierung von Netzwerken, Rechenzentren und deren Komponenten beschreibt die Abstraktion der physischen Hardware mit deren Eigenschaften durch virtuelle Objekte [10]. Diese virtuellen Objekte sind hardware-, hersteller- und teilweise auch technologieunabhängig, wodurch diese sehr variabel betrieben und migriert werden können. Diese zusätzliche Abstraktionsschicht separiert nun die physischen Netzwerke und deren Komponenten (z.B. Server, Switches und Ver-

bindungen), im Folgenden Substratnetzwerke und Substratelemente genannt, und die darauf betriebenen virtuellen Netzwerke und deren Komponenten, im Folgenden virtuelle Elemente genannt. Dabei stellt die Abstraktionsschicht Adapter zur Verfügung, um die virtuellen Elemente auf den Substratelementen zu betreiben. Dadurch können virtuelle Elemente wie Substratelemente behandelt und zusammengeschaltet werden, wodurch in virtuellen Servern wiederum eine Anzahl von virtuellen Prozessorkernen, Arbeits- oder Festplattenspeicher existiert. Dies ermöglicht es auf virtuellen Servern dieselbe Software (z.B. Betriebssystem oder Anwendungsprogramme) wie auf physischen Servern zu installieren, wodurch vorhandene Technologien, Arbeitsabläufe (z.B. Installations-, Wartungs- und Migrationsroutinen) und Methoden weiterhin Anwendung finden können.

Prinzipiell finden sich in der Literatur zwei Arten der Virtualisierung: die Aggregation oder das Aufsplitten von virtuellen Elementen [11]. Bei der Aggregation werden mehrere Substratelemente kombiniert und zusammen betrieben, um ein virtuelles Elemente abzubilden. Dadurch kann beispielsweise eine virtuelle Verbindung ein Zusammenschluss von mehreren Substratverbindungen mit dazwischenliegenden Switches oder Servern darstellen, welcher im Weiteren auch Substratpfad genannt wird. Wie dieser virtuelle Link bzw. Substratpfad tatsächlich im Substratnetzwerk umgesetzt wird, was auch die Konfiguration der dazwischenliegenden Switches oder Server beinhaltet, wird durch die Abstraktionsschicht vorgegeben. Genauso ist es möglich, mehrere Substratserver zu einem virtuellen Server zusammenzuschalten, wodurch Algorithmen nicht speziell auf die Nutzung von mehreren Servern zugeschnitten werden müssen, sondern die Verteilung der Daten-, Rechenleistung, sowie der notwendigen Kommunikation der Abstraktionsschicht überlassen wird. Das Aufsplitten von virtuellen Elementen auf mehrere Substratelemente stellt das Gegenstück zur Aggregation dar und ermöglicht es mehrere virtuelle Elemente auf einem Substratelement zu betreiben. Dabei gilt, wie zuvor auch, dass alle virtuellen Elemente vollständig unabhängig voneinander sind. So können beispielsweise mehrere virtuelle Server auf einem Substratserver betrieben werden, wodurch beispielsweise der physische Arbeitsspeicher in virtuelle Blöcke unterteilt zur Verfügung gestellt wird oder die physischen Prozessorkerne als einzelne virtuelle Komponenten oder in Zeitslots von gebündelten Prozessorkernen abgebildet werden können. Somit muss eine virtuelle Komponente wie ein Prozessorkern nicht notwendigerweise auch einem physischen Prozessorkern entsprechen, sondern kann auch ein Zeitfenster zur Nutzung des gesamten physischen Prozessors darstellen. Diese Praxis wird häufig bei Substratverbindungen angewendet, da hierbei mehrere virtuelle Verbindungen über eine Substratverbindung geführt werden können.

Gerade in Rechenzentren stellt die Virtualisierung eine Schlüsseltechnologie zum Betrieb dieser komplexen Umgebungen dar, um unterschiedliche Hardware herstellerunabhängig zu betreiben und verschiedene (virtuelle) Netzwerke und Topologien zu realisieren (z.B. ohne Verkabelungen anpassen zu müssen). Aber auch die Umsetzung von neuen Netzwerken, Technologien oder die Integration von automatisierten Arbeitsabläufen (z.B. Wartung oder Ausfall eines Servers, Migration von virtuellen Servern oder das Umleiten von Datenströmen) können durch eine einheitliche Schnittstelle und abstrahierte Sicht auf das Netzwerk rea-

lisiert werden. Somit können einfacher weitere Systeme (z.B. Abrechnungs- oder Einkaufssysteme) direkt mit eingebunden und eine integrierte Lösung des Betriebes eines kompletten Rechenzentrums erstellt werden.

2.2 Einbettung virtueller Netzwerke in Rechenzentren

In diesem Abschnitt beschreiben wir die Problemstellung zur Einbettung von virtuellen Netzwerken in Rechenzentren, die in dieser Arbeit verwendet wird. Zu Beginn definieren wir die Problemstellung informell und verdeutlichen diese anhand eines motivierenden Beispiels (Abschnitt 2.2.1). Danach stellen wir die Problemstellung sowohl als Graphrepräsentation (Abschnitt 2.2.2.1) als auch mathematische Formulierung mithilfe der ganzzahligen linearen Programmierung vor (Abschnitt 2.2.2.2). Anschließend geben wir einen Literaturüberblick über bestehende Algorithmen und Methoden zur Lösung der vorliegenden Problemstellung (Abschnitt 2.2.3).

2.2.1 *Grundlegende Problembeschreibung*

Die Problemstellung zur Einbettung von virtuellen Netzwerken in ein Substratnetzwerk, engl. Virtual Network Embedding (VNE), kurz *VNE-Problem*, stellt ein NP-hartes Optimierungsproblem [2] aus dem Bereich der Ressourcenallokation dar, um virtuelle Netzwerke möglichst effizient oder kostengünstig in einem Substratnetzwerk zu platzieren [28]. Das Ziel dieser Problemstellung liegt im Allgemeinen darin, die existierende (physische) Hardware durch eine dynamische Platzierung der virtuellen Elemente möglichst effizient und kostengünstig zu betreiben. Dabei kann das Substratnetzwerk auch selbst wieder als virtuelles Netzwerk angesehen werden, wodurch verschachtelte Virtualisierungen möglich sind und nur die niedrigste Ebene die physische Hardware beinhaltet. In der vorliegenden Arbeit wird als Anwendungsfall für das VNE-Problem eine Einbettung von virtuellen Netzwerken in ein Rechenzentrum betrachtet, welches als physisches Substratnetzwerk angenommen wird.

Daraus ergeben sich Anforderungen, die sowohl durch die physische Hardware als auch durch anwendungs-, kunden- oder bereichsspezifische Anforderungen festgelegt sind. Diese Anforderungen können in drei Kategorien unterteilt werden: (i) Knoten-, (ii) verbindungs- und (iii) anwendungsspezifische Anforderungen. *Knotenspezifische Anforderungen* (i) beschreiben hierbei Anforderungen, die durch die Hardwareelemente, im vorliegenden Fall *Server* und *Switches*, vorgegeben sind. Sie legen fest wie viele Hardwareressourcen zur Verfügung stehen (z.B. Prozessorkerne, Arbeits- oder Festplattenspeicher) oder welche zusätzlichen Funktionalitäten vorhanden sind (z.B. Hochleistungsprozessoren mit hoher Taktrate, Grafikkarten oder Verschlüsselungschips) und stellen sicher, dass diese Ressourcen nicht überbucht und nur vorhandene Funktionalitäten auch genutzt werden können. In dieser Arbeit werden wir uns bei Servern auf die Ressourcen *Prozessorkerne*, wobei zusätzlich noch zwischen *Hochleistungs-* und gängigen Serverprozessoren unterschieden wird, und *Arbeits-* und *Festplattenspeicher* beschränken, da diese *Ressourcen* auch in vielen Arbeiten die physischen Eigenschaften

repräsentieren [118, 79, 41, 120, 109]. Bei den Switches werden wir keine weiteren Anforderungen betrachten, da wir davon ausgehen, dass die Bandbreite und Verzögerungszeit durch die verwendeten Verbindungen und nicht durch die Switches bestimmt wird. Wir legen auch fest, dass virtuelle Server nur auf Substratservern, aber virtuelle Switches sowohl auf Substratswitches als auch auf Substratservern betrieben werden können, da Substratserver sowohl virtuelle Server sowie Switch-Funktionalitäten zur Verfügung stellen. Die Platzierung eines virtuellen Servers oder Switches A auf einen Substratserver oder -switch B wird dabei durch $A \rightarrow B$ ausgedrückt. Somit ergeben sich die folgenden knotenspezifischen Anforderungen:

K1 Jeder virtuelle oder Substratknoten ist entweder ein Server mit den Ressourcen Prozessorkerne, Arbeits- und Festplattenspeicher oder ein Switch.

K2 Jeder virtuelle Server (oder Switch) muss genau auf einem Substratserver (-server oder -switch) platziert werden.

K3 Die Summe von allen virtuellen Ressourcen (Prozessorkerne, Arbeits- oder Festplattenspeicher), die auf einen Substratknoten abgebildet werden, darf die jeweiligen Ressourcen dieses Substratknotens nicht übersteigen.

Daneben gibt es noch die *verbindungsspezifischen Anforderungen* (ii), welche die Anforderungen der physischen Verbindungen repräsentieren und somit deren zur Verfügung stehenden Hardwareressourcen (z.B. Bandbreite oder Verzögerungszeiten) oder Funktionalitäten (z.B. kabelgebunden oder Verschlüsselung) festlegen und sicherstellen, dass diese Ressourcen nicht überbucht oder nur vorhandene Funktionalitäten genutzt werden. In dieser Arbeit werden wir uns auf die *Bandbreite* bei den Verbindungen beschränken, da diese den wichtigsten Einfluss bei der Platzierung darstellt und auch in vielen Arbeiten als *Ressource* für Verbindungen herangezogen wird [76, 41, 120, 109]. Zusätzlich muss sichergestellt werden, dass die Start- und Zielknoten der virtuellen Verbindung auch auf die Start- und Zielknoten des Substratpfades abgebildet sind. Im Weiteren werden Verbindungen zwischen dem Startknoten A und dem Zielknoten B durch A-B repräsentiert. Hieraus ergeben sich die folgenden verbindungsspezifischen Anforderungen:

V1 Jede virtuelle oder Substratverbindung besitzt die Ressource Bandbreite.

V2 Jede virtuelle Verbindung muss genau auf einem azyklischen Substratpfad platziert werden, wobei die jeweiligen Start- und Zielknoten der virtuellen Verbindung auf den Start- und Zielknoten des Substratpfades platziert sein müssen.

V3 Die Summe von allen virtuellen Ressourcen (Bandbreite), die auf eine Substratverbindung abgebildet werden, darf die jeweiligen Ressourcen (Bandbreite) der Substratverbindung nicht übersteigen. Die Bandbreite innerhalb eines Substratservers wird dabei als unbegrenzt angesehen.

Die letzte Kategorie der *anwendungsfallspezifischen Anforderungen* (iii) kann sowohl hardwarebedingte Anforderungen als auch funktionale oder nicht-funk-

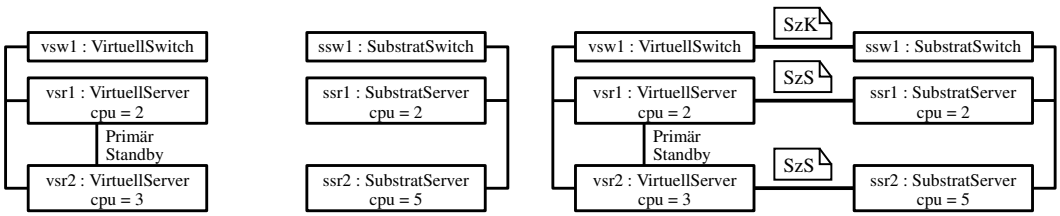
tionale Anforderungen beschreiben. Diese Anforderungen können aus anwendungsspezifischen Anforderungen (z.B. festgelegte Anwendungsketten oder verteilte/parallele Anwendungen), kundenspezifischen Anforderungen (z.B. kurze Verzögerungszeiten, Backupkonzepte oder Service-Level-Agreements) oder betriebsspezifischen Anforderungen (z.B. Sicherheitsrichtlinien oder internen Arbeitsabläufen) bestehen. In dieser Arbeit werden wir uns exemplarisch nur auf die beiden Anforderungen für Ausfallsicherungen für Server und kurze Verzögerungszeiten konzentrieren. Eine *Ausfallsicherung* für einen primären Server definiert dazu einen Standby-Server, der im Fehlerfall des primären Servers (z.B. Ausfall oder Wartung) automatisch dessen Funktion übernimmt. Damit diese Ausfallsicherung auch im Falle eines Serverausfalls im Substratnetzwerk gewährleistet wird, muss sichergestellt werden, dass die beiden virtuellen Server nicht auf demselben Substratserver betrieben werden. Die Anforderung an eine *kurze Verzögerungszeit* wird in dieser Arbeit dadurch realisiert, dass alle virtuellen Links in einem virtuellen Netzwerk, welches eine kurze Verzögerungszeit fordert, nur auf Substratpfade mit einer maximalen Länge von zwei abgebildet werden dürfen. In den Netzwerken im Rechenzentren bedeutet dies im Allgemeinen, dass die Kommunikation nur innerhalb eines Racks stattfindet. Daraus ergeben sich die folgenden anwendungsfallspezifischen Anforderungen:

A1 (Ausfallsicherheit) Jeder virtuelle primäre Server darf maximal einen virtuellen Standby-Server besitzen, wobei diese beiden Server nicht identisch sein dürfen.

A2 (Ausfallsicherheit) Der virtuelle primäre und Standby-Server dürfen nicht auf demselben Substratserver platziert werden.

A3 (Kurze Verzögerungszeit) Wenn das virtuelle Netzwerk eine kurze Verzögerungszeit fordert, dann darf jede virtuelle Verbindung nur auf einem Substratpfad mit maximal zwei Substratverbindungen abgebildet werden.

Da das VNE-Problem ein Optimierungsproblem darstellt, wird neben den Anforderungen auch noch eine *Zielfunktion* definiert, die maximiert oder minimiert wird. Diese Zielfunktionen sind sehr unterschiedlich gestaltet, wobei gängige Zielfunktionen die Minimierung der Kommunikations-, der monatlichen Betriebskosten oder der verwendeten Hardware darstellen [28, 118].



(a) Ein virtuelles und Substratnetzwerk **(b)** Lösungsmöglichkeit

Abbildung 2.3: Motivierendes Beispiel für das VNE-Problem

Tabelle 2.1: Überprüfung der Anforderungen für die Lösung aus Abb. 3.3a

Anf.	Erfüllt	Anmerkungen
K1	✓	Es existieren nur Server und Switches in den Netzwerken
K2	✓	*vsw1*→*ssw1*, *vsr1*→*ssr1* und *vsr2*→*ssr2*
K3	✓	$vsr1.cpu \leq ssr1.cpu$ ($2 \leq 2$) und $vsr2.cpu \leq ssr2.cpu$ ($3 \leq 5$)
V1	✓	Verbindung *vsw1-vsr1*→ *ssw1-ssr1* und *vsw1-vsr2*→ *ssw1-ssr2*
V2	✓	Bandbreiten werden in diesem Beispiel nicht beachtet
A1	✓	Der primäre Server *vsr1* hat einen Standby-Server *vsr2*
A2	✓	*vsr1*→*ssr1* und *vsr2*→*ssr2*
A3	✓	Keine kurze Verzögerungszeit gefordert

Beispiel 2.1 (Motivierendes Beispiel)**.** Im Folgenden wird das motivierende Beispiel für diese Arbeit vorgestellt. Dieses Beispiel dient als Ausgangspunkt zur Veranschaulichung des VNE-Problems und den vorgestellten Methoden und Techniken.

Ausgangssituation Abbildung 2.3a zeigt beispielhaft ein virtuelles Netzwerk auf der linken Seite und ein Substratnetzwerk auf der rechten Seite, welche sternförmig mit jeweils einem zentralen Switch (*vsw1* bzw. *ssw1*) und je zwei Servern (*vsr1*, *vsr2* bzw. *ssr1*, *ssr2*) verbunden sind. Die virtuellen Server *vsr1* und *vsr2* benötigen 2 bzw. 3 Prozessorkerne (*cpu*) und die Substratserver *ssr1* und *ssr2* können 2 bzw. 5 Prozessorkerne (*cpu*) zur Verfügung stellen. Dabei stellt der virtuelle Server *vsr2* eine Ausfallsicherung für den primären Server *vsr1* dar, weshalb diese beiden virtuellen Server nicht auf demselben Substratserver platziert werden dürfen. Alle weiteren Ressourcen für Server und auch die Ressourcen für die virtuellen und Substratverbindungen werden in diesem Beispiel zur Vereinfachung nicht betrachtet.

Mögliche Lösung des VNE-Problems Eine mögliche Lösung des VNE-Problems aus Abb. 2.3a stellt Abb. 3.3a dar, wobei beispielhaft eine Zielfunktion zur Minimierung der Kommunikationskosten zwischen den einzelnen virtuellen Server gewählt wurde. Bei dieser Lösung wird der virtuelle Switch *vsw1* auf den Substratswitch *ssw1* (*vsw1*→*ssw1*), der virtuelle Server *vsr1* auf den Substratserver *ssr1* (*vsr1*→*ssr1*) und *vsr2* auf *ssr2* (*vsr2*→*ssr2*) abgebildet. Diese Lösung erfüllt alle vorher definierten Anforderungen, wie in der Tabelle 2.1 verdeutlicht wird.

2.2.2 *Problemdefinition*

In diesem Abschnitt stellen wir die Problemdefinition sowohl als Graphrepräsentation als auch als mathematische Formulierung vor.

2.2.2.1 *Darstellung der Problemdefinition als Graph*

Zur Definition der Substratnetzwerke und virtuellen Netzwerke wird die folgende Definition eines Netzwerkgraphen G verwendet.

Definition 2.1 (Netzwerkgraph)**.** Ein Netzwerkgraph $G = (N, L, n, C, M, S, B, T)$ wird als ungerichteter gewichteter Graph mit einer Menge von Knoten N, Verbindungen L, Anzahl an Verbindungen n, einer Menge von Ressourcen (Prozessorkerne C, Arbeitsspeicher M, Festplattenspeicher S und Bandbreite B) und einer Typisierung T (Server Sr oder Switch Sw) definiert. Dabei gilt folgendes:

$$
\begin{aligned}
&n : L \to 2^N, \text{ wobei gilt } \forall l \in L : |n(l)| = 2\\
&C, M, S : N \to \mathbb{N}_0\\
&B : L \to \mathbb{N}_0\\
&T : N \to \{Sr, Sw\}
\end{aligned}
$$

Ausgehend von Definition 2.1 ist das Substratnetzwerk, welches in dieser Arbeit das Netzwerk in einem Rechenzentrum darstellt, als $G^S = (N^S, L^S, n^S, C^S, M^S, S^S, B^S, T^S)$ definiert. Das hochgestellte S dient hierbei als Indikator für das Substratnetzwerk. Zusätzlich existieren noch Pfade P^S, welche als eine Sequenz von azyklisch miteinander verbundenen Verbindungen vom Knoten u zum Knoten v definiert sind.

$$
\begin{aligned}
P^S = \{&p_{u,v} = \{l_{1,2}, \ldots, l_{k-1,k}\} \subseteq L^S, \text{ mit } n_1 = u, n_k = v \in N^S,\\
&\exists \{n_i, \ldots, n_{i+1}\} \in N^S : \forall i \in 0, \ldots, k-1 : n(l) = \{n_i, n_{i+1}\},\\
&k \in \mathbb{N}_{>0}, \vee p_{u,v} = \{l_{u,v}\} \subseteq L^S\}
\end{aligned}
\tag{2.1}
$$

Die Länge eines Substratpfades $p_{u,v}$ gibt hierbei die Anzahl der in diesem Pfad enthaltenen Knoten an.

$$
länge(p_{u,v}) = |p_{u,v}| - 1 \tag{2.2}
$$

Analog zum Substratnetzwerk werden auch die virtuellen Netzwerke als $G^V = (N^V, L^V, n^V, C, M, S, B, T)$ festgelegt. Hierbei kennzeichnet das hochgestellte V ein virtuelles Netzwerk.

Eine Platzierung m eines virtuellen Netzwerkes G^V auf das Substratnetzwerk G^S (kurz $G^V \to G^S$) kann hierbei als Morphismus des Graphen G^P auf G^S dargestellt werden, wobei folgendes gilt:

$$
\begin{aligned}
G^P := &(N^P, L^P, n^P, C^P, M^P, S^P, B^P, T^P) \text{ mit}\\
&N^P := N^S, \mathbf{L^P} := \mathbf{P^S}, n^P := n^S, C^P := C^S,\\
&M^P := M^S, S^P := S^S, B^P := B^S, T^P := T^S\\
&M(p_{u,v}) = min_{l \in p_{u,v}}(M(l))
\end{aligned}
\tag{2.3}
$$

2.2.2.2 *Darstellung der Problemdefinition als ILP-Formulierung*

Zur Definition des VNE-Problems als mathematische Formulierung wird in dieser Arbeit die Methode der *ganzzahligen linearen Programmierung*, engl. Integer Linear

Programming (ILP) [88, 106], verwendet, welche in dieser Domäne eine etablierte Beschreibungssprache darstellt. Hierbei wird das VNE-Problem als eine Menge von unbekannten ganzzahligen Variablen, linearen Einschränkungen und einer linearen Zielfunktion beschrieben. Generell kann ein ILP-Problem in der folgenden Form definiert werden, wobei die Variablen nur die Werte 0 oder 1 annehmen dürfen.

$$max\{cx|Ax \leq b; x \in \{0,1\}\} \text{ [88].} \tag{2.4}$$

Dabei beschreibt x einen Vektor für die ganzzahligen Variablen mit dem Wertebereich {0,1}, A eine rationale Matrix und b und c rationale Vektoren der passenden Dimension. Hierbei wird für die Menge aller Variablen x eine Lösung gefunden, die alle Einschränkungen $Ax \leq b$ einhalten und ein Maximum von cx darstellen.

Im Folgenden wird nun die Definition des VNE-Problems als ILP-Formulierung für diese Arbeit vorgestellt, welche durch [85, 118, 96] inspiriert wurde.

Substratnetzwerk Das Substratnetzwerk, welches ein Netzwerk in einem Rechenzentrum modelliert, besteht aus einer Menge von Knoten N^S und Verbindungen L^S. Auch hier dient das hochgestellte S als Indikator für das Substratnetzwerk. Zusätzlich existiert noch eine Menge von azyklischen Pfaden P^S (siehe Anforderung V2), wobei ein Pfad von n_1 zu n_k als $p_{n_1,n_k} = (n_1, \ldots, n_k)$ festgelegt ist. Die Länge $länge(p_{u,v})$ eines Substratpfades wird als die Anzahl der in diesem Pfad enthaltenen Knoten definiert.

$$länge(p_{u,v}) = |p_{u,v}| - 1 \tag{2.5}$$

Die Knoten $u \in N^S$ können zusätzlich als Server mit den Ressourcen Prozessorkerne C_u, Arbeitsspeicher M_u und Festplattenspeicher S_u oder Switch festgelegt werden (siehe Anforderung K1). Verbindungen $l_{u,v} \in L^S$, welche vom Startknoten u zum Zielknoten v definiert sind, besitzen hingegen die Ressource Bandbreite $B_{l_{u,v}}$ (siehe Anforderung V1).

$$\begin{aligned} \forall u \in N^S &: C_u, M_u, S_u \in \mathbb{N}^+ \\ \forall l_{u,v} \in L^S &: B_{l_{u,v}} \in \mathbb{N}^+, u, v \in N^S \end{aligned} \tag{2.6}$$

Die Definition des Typs eines Knotens u (Server Sr oder Switch Sw) legt hierbei fest, welcher virtuelle Knoten auf diesem Substratknoten u platziert werden darf (siehe Anforderung K2). Somit darf sowohl ein virtueller Server als auch ein virtueller Switch auf einem Substratserver platziert werden, aber nur ein virtueller Switch auf einem Substratswitch.

$$\forall u \in N^S : \begin{cases} Sr_u, Sw_u \in \{0,1\} \\ Sr_u = 1, \text{ auf } u \text{ kann ein vir. Switch oder Server platziert werden} \\ Sw_u = 1, \text{ auf } u \text{ kann nur ein vir. Switch platziert werden} \end{cases} \tag{2.7}$$

Beispiel 2.2 (ILP Beschreibung des Substratnetzwerks)**.** Das Substratnetzwerk aus Abbildung 2.3a besteht aus den Knoten (N^S) *ssw1*, *ssr1* und *ssr2*, den Verbindungen (L^S) *ssw1-ssr1* und *ssw1-ssr2* und den Pfaden (P^S) *ssw1-ssr1*,

ssw1-ssr2 und *ssr1-ssr2*, wobei manche Pfade den Verbindungen entsprechen. In diesem Beispiel besitzt Server *ssr1* zwei Prozessorkerne ($C_{ssr1}.cpu = 2$) und *ssr2* fünf Prozessorkerne ($C_{ssr2}.cpu = 5$). Die übrigen Ressourcen (Arbeits-, Festplattenspeicher und Bandbreite) werden zur Verbesserung der Übersichtlichkeit nicht berücksichtigt. Da *ssw1* einen Switch repräsentiert, gilt zusätzlich $Sw_{ssw1} = 1$ und $Sr_{ssw1} = 0$. Bei den beiden Servern *ssr1* und *ssr2* ergeben sich folgende Gleichungen: $Sw_{ssr1} = 1$, $Sr_{ssr1} = 1$, $Sw_{ssr2} = 1$ und $Sr_{ssr2} = 1$.

Virtuelle Netzwerke Analog zum Substratnetzwerk besitzen die virtuellen Netzwerke die Knoten N^V und Verbindungen L^V, wobei hier das hochgestellte V ein virtuelles Netzwerk kennzeichnet. Da die Ressourcen in den virtuellen und im Substratnetzwerk identisch definiert sind (siehe Anforderungen K1 und V1), besitzt ein virtueller Server $i \in N^V$ auch die Ressourcen Prozessorkerne C_i, Arbeitsspeicher M_i und Festplattenspeicher S_i, ein virtueller Switch keine weiteren Ressourcen und eine virtuelle Verbindung $l_{i,j} \in L^V$ zwischen dem Startknoten i und dem Zielknoten j die Ressource Bandbreite $B_{l_{i,j}}$. Lediglich die Interpretation dieser Ressourcen unterscheidet sich zum Substratnetzwerk, da in diesem Fall die geforderten Ressourcen beschrieben werden, wo hingegen im Substratnetzwerk die zur Verfügung stehenden Ressourcen modelliert sind. Bei der Abbildung der virtuellen Funktionalität (Server oder Switch) ergibt sich allerdings ein Unterschied zum Substratnetzwerk, da ein virtueller Knoten $i \in N^V$ nur genau eine Funktionalität (Server oder Switch) besitzen kann, wo hingegen auf einem Substratknoten (z.B. Substratserver) auch beide virtuelle Funktionalitäten (z.B. Server und Switch) platziert werden können.

$$\forall i \in N^V : \begin{cases} Sr_i, Sw_i \in \{0,1\} \\ Sr_i + Sw_i = \{0,1\} \\ Sr_i = 1, i \text{ ist ein Server} \\ Sw_i = 1, i \text{ ist ein Switch} \end{cases} \tag{2.8}$$

Zusätzlich können virtuelle Server noch eine Ausfallsicherheit in Form eines Standby-Server besitzen (Anforderung A1). Diese Eigenschaft wird durch $f_{i,j} \in \{0,1\}$ modelliert, wobei der Knoten i als primär und j als Standby angesehen wird. Dabei kann eine Ausfallsicherheit aber nur zwischen zwei Servern bestehen, wodurch Beziehungen zwischen Servern und Switches oder Switches und Switches implizit durch 0 festgelegt sind.

$$\forall i \in N^V, \forall j \in N^V : \begin{cases} f_{i,j} \in \{0,1\} \\ f_{i,j} = 0, \text{ ohne Ausfallsicherung } \vee Sw_i = 1 \vee Sw_j = 1 \\ f_{i,j} = 1, \text{ mit Ausfallsicherung } \wedge Sr_i = 1 \wedge Sr_j = 1 \end{cases} \tag{2.9}$$

In Tabelle 2.2 werden die eingeführten und zur Laufzeit konstanten ILP-Variablen nochmal zusammengefasst.

Tabelle 2.2: Zur Laufzeit konstante ILP-Variablen für die Netzwerke

Bezeichung	Bereich	Beschreibung
$l_{u,v}$	$\{0,1\}$	Verbindung $l_{u,v}$ zwischen dem Start- u und dem Zielknoten v mit $u, v \in N$
$p_{u,v}$	$\{0,1\}$	Substratpfad $p_{u,v}$ zwischen dem Start- u und dem Zielknoten v mit $u, v \in N$
$länge(p_{u,v})$	$\mathbb{N}$	Länge eines Substratpfades $p_{u,v}$ zwischen dem Start- u und dem Zielknoten v mit $u, v \in N$
C_u, M_u, S_u	$\mathbb{N}$	Prozessorkerne, Arbeits- und Festplattenspeicher eines Servers $u \in N$
$B_{l_{u,v}}$	$\mathbb{N}$	Bandbreite einer Verbindung $l_{u,v} \in L$
Sr_u, Sw_u	$\{0,1\}$	Platzierung eines virtuellen Servers/Switches möglich

Beispiel 2.3 (ILP Beschreibung des virtuellen Netzwerks)**.** Das virtuelle Netzwerk aus Abbildung 2.3a besteht, analog zum Substratnetzwerk, aus den Knoten (N^V) *vsw1*, *vsr1* und *vsr2* und den Verbindungen (L^V) *vsw1-vsr1* und *vsw1-vsr2*. Die beiden Server *vsr1* und *vsr2* besitzen je zwei bzw. drei Prozessorkerne ($C_{vsr1}.cpu = 2$ und $C_{vsr2}.cpu = 3$). Da *vsw1* einen Switch repräsentiert, gilt zusätzlich $Sw_{vsw1} = 1$ und $Sr_{vsw1} = 0$. Bei den beiden Servern *vsr1* und *vsr2* ergibt sich Folgendes: $Sw_{vsr1} = 1$, $Sr_{vsr1} = 1$, $Sw_{vsr2} = 1$ und $Sr_{vsr2} = 1$. Zusätzlich ist *vsr2* ein Standby-Server für den primären Server *vsr1*, wodurch folgende Beziehungen gelten:

$$f_{vsr1,vsr2} = 1$$
$$f_{vsr2,vsr1} = 0$$
$$f_{vsw1,vsr1} = 0$$
$$f_{vsr1,vsw1} = 0$$
$$[\dots]$$

Platzierungsvariablen Zur Modellierung der eigentlichen Platzierungen oder Platzierungskandidaten werden binäre Variablen (Platzierungsvariablen) mit dem Wertebereich $\{0,1\}$ verwendet. Da bei der Platzierung virtuelle Knoten nur auf Substratknoten (siehe Anforderung K2) und virtuelle Verbindungen nur auf Substratpfade (siehe Anforderung V2) abgebildet werden dürfen, werden zwei Mengen von Platzierungsvariablen zur Repräsentation dieser Eigenschaften verwendet. Die Platzierungsvariablen für die Knotenplatzierungen x_u^i repräsentieren die Abbildung eines virtuellen Knotens i auf einen Substratknoten u, wodurch eine Variable für jede Kombination von virtuellen und Substratknoten existiert. Die Platzierung eines virtuellen Knotens i auf einen Substratknoten u wird somit durch $x_u^i = 1$ repräsentiert.

$$\forall i \in N^V : \forall u \in N^S : x_u^i \in \{0,1\} \tag{2.10}$$

Tabelle 2.3: ILP-Variablen für die Platzierung (Platzierungsvariablen) eines virtuellen Netzwerks auf ein Substratnetzwerk

Bezeichung	Bereich	Beschreibung
x_u^i	$\{0,1\}$	Platzierungsvariable eines virtuellen Knotens $i \in N^V$ auf einem Substratknoten $u \in N^S$
$y_{p_{u,v}}^{l_{i,j}}$	$\{0,1\}$	Platzierungsvariable einer virtuellen Verbindung $l_{i,j} \in L^V$ auf einem Substratpfad $p_{u,v} \in P^S$

Analog zu den Platzierungsvariablen für die Knotenplatzierung werden die Variablen für die Verbindungsplatzierung modelliert. Somit gibt es für jede Kombination von virtuellen Verbindungen zu Substratpfaden eine Variable $y_{p_{u,v}}^{l_{i,j}}$, welche für $y_{p_{u,v}}^{l_{i,j}} = 1$ die virtuelle Verbindung $l_{i,j}$ auf den Substratpfad $p_{u,v}$ vorsieht.

$$\forall l_{i,j} \in L^V : \forall p_{u,v} \in P^S : y_{p_{u,v}}^{l_{i,j}} \in \{0,1\} \tag{2.11}$$

In Tabelle 2.3 werden die Platzierungsvariablen nochmal zusammengefasst.

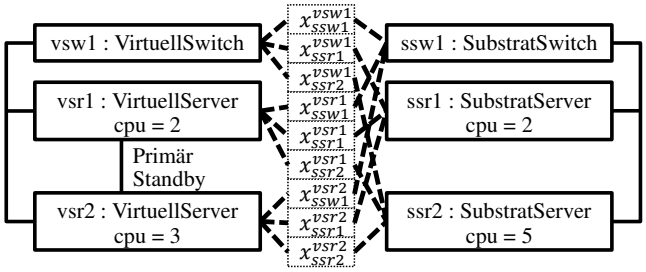


Abbildung 2.4: ILP Platzierungsvariablen für die Knotenplatzierungen

Beispiel 2.4. In Abb. 2.4 werden die Platzierungsvariablen für die Knotenplatzierungen dargestellt. Jede Platzierungsvariable x_u^i (z.B. x_{ssr1}^{vsw1}) beschreibt eine mögliche Platzierung bzw. einen Platzierungskandidaten (z.B. *vsw1* auf *ssr1*), wodurch für jede Kombination von virtuellem zu einem Substratknoten eine Platzierungsvariable existiert.

2.2.2.3 *Einschränkungen*

Nachdem nun die Netzwerke und die Platzierungsvariablen definiert wurden, legen wir in diesem Abschnitt die Einschränkungen fest, die sich durch die knoten-, verbindungs- und anwendungsfallspezifischen Anforderungen ergeben. Diese Anforderungen werden hierbei als lineare (Un-)Gleichungen definiert.

Knotenspezifische Anforderungen Die knotenspezifischen Anforderungen (siehe Anforderungen K1 bis K3) legen die Bedingungen für eine Platzierung eines virtuellen auf einen Substratknoten fest. Dazu wird in Anforderung K1 gefordert, dass jeder Knoten entweder einen Server (mit den Ressourcen Prozessorkerne, Arbeits- und Festplattenspeicher) oder einen Switch darstellt. Diese Anfor-

derung wurde bereits in der Modellierung des virtuellen und des Substratnetzwerks berücksichtigt.

In Anforderung K2 wird zusätzlich gefordert, dass jeder virtuelle Server (Switch) auf genau einem Substratserver (-server oder -switch) platziert wird. Sie enthält somit implizit zwei Forderungen: (i) Jeder virtuelle Knoten muss genau auf einen Substratknoten und (ii) ein virtueller Server (Switch) darf nur auf einem Substratserver (-server oder -switch) platziert werden. Die erste Forderung (i) bedeutet somit, dass wenn ein virtueller Knoten $i \in N^V$ auf einem Substratknoten $u \in N^S$ platziert wurde ($x_u^i = 1$), keine andere Platzierungsvariable x_v^i für den virtuellen Knoten i und den Substratknoten $v \neq u \in N^S$ ausgewählt wird ($x_v^i = 1$).

$$\forall i \in N^V : \sum_{u \in N^S} x_u^i = 1 \tag{2.12}$$

Hierbei ist $\sum_{u \in N^S} x_u^i$ definiert als die Summe über alle Platzierungsvariablen x_1^i, $\ldots$, x_k^i eines virtuellen Knotens i und k Substratknoten.

Beispiel 2.5 (Gleichungen für Anforderung K2 (i))**.** Damit in Abb. 2.4 jeder virtuelle Knoten nur auf einen Substratknoten abgebildet wird, muss für jeden virtuellen Knoten (*vsw1*, *vsr1* und *vsr2*) die Summe über alle Platzierungsvariablen x_u^i (siehe Gleichung 2.12) gebildet werden. Somit müssen folgende Gleichungen erzeugt und bei der Lösung mit beachtet werden:

$$\begin{aligned}
\text{Für } vsw1 &\Rightarrow x_{ssw1}^{vsw1} + x_{ssr1}^{vsw1} + x_{ssr2}^{vsw1} = 1 \\
\text{Für } vsr1 &\Rightarrow x_{ssw1}^{vsr1} + x_{ssr1}^{vsr1} + x_{ssr2}^{vsr1} = 1 \\
\text{Für } vsr2 &\Rightarrow x_{ssw1}^{vsr2} + x_{ssr1}^{vsr2} + x_{ssr2}^{vsr2} = 1
\end{aligned}$$

Die zweite Forderung (ii) kann durch logische Implikationen ausgedrückt werden. Als Beispiel wird der Fall betrachtet, dass ein virtueller Server i auf einem Substratknoten u platziert wird ($Sr_i = 1$ und $x_u^i = 1$). Dies impliziert, dass der Substratknoten u einen Server darstellt ($Sr_u = 1$). Da $a \rightarrow b$ im Wertbereich {0,1} äquivalent zu $a \leq b$ ist (siehe [44] T1.8), folgt daraus folgende Beziehung: $Sr_i x_u^i \leq Sr_u$. Hierbei wird der $\leq$-Operator zur Beschreibung der logischen Implikation verwendet [44, 63]. Dabei besitzen Sr_i und Sr_u zur Laufzeit konstante Werte, wodurch die Linearität dieser Ungleichung sichergestellt wird.

$$\forall i \in N^V : \forall u \in N^S : Sr_i x_u^i \leq Sr_u, Sw_i x_u^i \leq Sw_u + Sr_u \tag{2.13}$$

Beispiel 2.6 (Ungleichungen für Anforderung K2 (ii))**.** Damit in Abb. 2.4 ein virtueller Server immer nur auf einem Substratserver und ein virtueller Switch auf einen Substratserver oder -switch abgebildet wird, müssen die Ungleichungen, welche sich aus Gleichung 2.13 ergeben, beachtet werden. Für *vsw1* sind dies folgende Ungleichungen:

$$\begin{aligned}
Sr_{vsw1} x_{ssw1}^{vsw1} \leq Sr_{ssw1}, \quad Sw_{vsw1} x_{ssw1}^{vsw1} &\leq Sw_{ssw1} + Sr_{ssw1} \\
Sr_{vsw1} x_{ssr1}^{vsw1} \leq Sr_{ssr1}, \quad Sw_{vsw1} x_{ssr1}^{vsw1} &\leq Sw_{ssr1} + Sr_{ssr1} \\
Sr_{vsw1} x_{ssr2}^{vsw1} \leq Sr_{ssr2}, \quad Sw_{vsw1} x_{ssr2}^{vsw1} &\leq Sw_{ssr2} + Sr_{ssr2}
\end{aligned}$$

Für *vsr1* sind dies folgende Ungleichungen:

$$Sr_{vsr1}x^{vsr1}_{ssw1} \leq Sr_{ssw1}, \quad Sw_{vsr1}x^{vsr1}_{ssw1} \leq Sw_{ssw1} + Sr_{ssw1}$$
$$Sr_{vsr1}x^{vsr1}_{ssr1} \leq Sr_{ssr1}, \quad Sw_{vsr1}x^{vsr1}_{ssr1} \leq Sw_{ssr1} + Sr_{ssr1}$$
$$Sr_{vsr1}x^{vsr1}_{ssr2} \leq Sr_{ssr2}, \quad Sw_{vsr1}x^{vsr1}_{ssr2} \leq Sw_{ssr2} + Sr_{ssr2}$$

Durch Anforderung K3 wird festgelegt, dass die Ressourcen jedes Substratknotens nicht durch die darauf platzierten virtuellen Ressourcen überbucht werden dürfen. Dazu werden die angeforderten Ressourcen eines virtuellen Servers i (z.B. Prozessorkerne C_i) als Koeffizienten für die dazugehörigen Platzierungsvariablen x^i_u ($u \in N^S$) betrachtet und gehen hierbei nur in die Summe der benötigten Substratressourcen ein, wenn eine Platzierung stattfindet ($x^i_u = 1$). Somit ergeben sich die folgenden zusätzlichen Ungleichungen:

$$\forall u \in N^S : \sum_{i \in N^V} C_i x^i_u \leq C_u \qquad (\text{ILP}_{\text{cpu}})$$
$$\forall u \in N^S : \sum_{i \in N^V} M^V_i x^i_u \leq M_u \qquad (\text{ILP}_{\text{ram}})$$
$$\forall u \in N^S : \sum_{i \in N^V} S_i x^i_u \leq S_u \qquad (\text{ILP}_{\text{hdd}})$$

Beispiel 2.7 (Ungleichungen für Anforderung K3)**.** Damit in Abb. 2.4 nun auch noch alle virtuellen und Substratressourcen beachtet werden, ergeben sich für die Prozessorkerne C folgende Ungleichungen:

$$\text{Für } ssr1 \Rightarrow C_{vsw1}x^{vsw1}_{ssr1} + C_{vsr1}x^{vsr1}_{ssr1} + C_{vsr2}x^{vsr2}_{ssr1} \leq C_{ssr1}$$
$$\text{Für } ssr2 \Rightarrow C_{vsw1}x^{vsw1}_{ssr2} + C_{vsr1}x^{vsr1}_{ssr2} + C_{vsr2}x^{vsr2}_{ssr2} \leq C_{ssr2}$$

Verbindungsspezifische Anforderungen Bei den verbindungsspezifischen Anforderung (siehe Anforderungen V1 bis V3) wird festgelegt, dass jeder virtuelle Link auf einen Substratpfad platziert wird und die geforderten Ressourcen (Bandbreite) auch zur Verfügung gestellt bekommt. Dazu wird in Anforderung V1 festgelegt, dass jede virtuelle und Substratverbindung die Ressource Bandbreite besitzt, wobei diese Anforderung schon bei der Modellierung der Netzwerke beachtet wurde. In Anforderung V2 sind, wie auch bei Anforderung K2, mehrere Forderungen enthalten: (i) Jeder virtuelle Link muss auf genau einen Substratpfad abgebildet werden, (ii) jeder Substratpfad darf keine Zyklen beinhalten und (iii) der Start- und Zielknoten des virtuellen Links muss auf dem jeweiligen Start- und Zielknoten des Substratpfades platziert werden. Forderung (i), dass ein virtueller Link $l_{i,j} \in L^V$ genau auf einen Substratpfad $p_{u,v} \in P^S$ platziert wird, kann analog zur Anforderung K2 Gleichung 2.12 umgesetzt werden.

$$\forall l_{i,j} \in L^V : \sum_{p_{u,v} \in P^S} y^{l_{i,j}}_{p_{u,v}} = 1 \qquad (2.14)$$

Zur Einhaltung von Forderung (ii) werden die Platzierungsvariablen für die Knoten- und Verbindungsplatzierung miteinander in Beziehung gesetzt. Dadurch kann

sichergestellt werden, dass wenn der virtueller Link $l_{i,j}$ auf einen Substratpfad $p_{u,v}$ platziert wird ($y_{p_{u,v}}^{l_{i,j}} = 1$) auch der virtuelle Startknoten i (Zielknoten j) auf dem Startknoten des Substratpfades u (Zielknoten v) platziert ist ($x_u^i = 1$ bzw. $x_v^j = 1$). Diese logische Implikation wird wiederum durch den $\leq$-Operator definiert.

$$\forall l_{i,j} \in L^V : \forall p_{u,v} \in P^S : y_{p_{u,v}}^{l_{i,j}} \leq x_u^i \tag{2.15}$$

$$\forall l_{i,j} \in L^V : \forall p_{u,v} \in P^S : y_{p_{u,v}}^{l_{i,j}} \leq x_v^j \tag{2.16}$$

Die Einhaltung der Anforderung an die Bandbreitenbeschränkungen für Verbindungen (siehe Anforderung V3) erfolgt analog zur Ressourcenanforderung bei Knoten. Hierbei werden die geforderten Bandbreiten der virtuellen Verbindungen auch wieder als Koeffizienten für die Platzierungsvariablen $y_{p_{u,v}}^{l_{i,j}}$ verwendet, wodurch sie nur in die Summe der benötigen Bandbreite B_e für die Substratverbindungen e im Substratpfad $p_{u,v}$ eingehen, wenn eine Platzierung erfolgt ($y_{p_{u,v}}^{l_{i,j}} = 1$). Da hierbei mehrere Substratpfade $p_{u,v}$ über eine Substratverbindungen e laufen können, müssen die Platzierungsvariablen von allen virtuellen Verbindungen $l_{i,j}$ auf alle Substratpfade $p_{u,v}$ für die Substratverbindungen e mit den jeweiligen Bandbreiten der virtuellen Verbindung $l_{i,j}$ aufsummiert werden.

$$\forall e \in L^S : \sum_{\substack{p_{u,v} \in P^S,\, l_{i,j} \in L^V \\ e \in p_{u,v}}} B_{l_{i,j}} y_{p_{u,v}}^{l_{i,j}} \leq B_e \tag{ILP$_{\text{bb}}$}$$

Anwendungsspezifische Anforderungen Bei den anwendungsfallspezifischen Anforderungen unterscheiden wir zwischen Anforderungen bezüglich der Ausfallsicherheit (siehe Anforderungen A1 und A2) und der Verzögerungszeit (siehe Anforderung A3).

Bei der Ausfallsicherheit wird zur Laufzeit festgelegt, ob ein virtueller primärer Server einen virtuellen Standby-Server besitzt, wobei einem primäre Server maximal ein Standby-Server zugewiesen werden darf, wobei der primäre und der Standby-Server nicht derselbe Server sein dürfen (siehe Anforderung A1).

$$\begin{aligned} &\forall i \in N^V : f_{i,i} = 0 \\ &\forall i \in N^V : \sum_{j \in N^V, i \neq j} f_{i,j} \leq 1 \end{aligned} \tag{2.17}$$

Beispiel 2.8 (Ungleichungen für Anforderung A1)**.** Damit in Abb. 2.4 auch die Anforderung A1 bezüglich der Ausfallsicherheit, dass für jeden primären Server maximal ein Standby-Server existieren darf, eingehalten wird, werden zusätzliche Einschränkungen nach Gleichung 2.17 benötigt. So wird durch die folgenden Gleichungen sichergestellt, dass der primäre und der Standby-Server nicht ein und derselbe Server sind.

$$\begin{aligned} f_{vsw1,vsw1} &= 0 \\ f_{vsr1,vsr1} &= 0 \\ f_{vsr2,vsr2} &= 0 \end{aligned}$$

Um nun sicherzustellen, dass jeder primäre Server maximal einen Standby-Server besitzt, werden die folgenden Ungleichungen benötigt.

$$f_{vsw1,vsr1} + f_{vsr1,vsr2} \leq 1$$
$$f_{vsr1,vsw1} + f_{vsr1,vsr2} \leq 1$$
$$f_{vsr2,vsw1} + f_{vsr2,vsr1} \leq 1$$

Bei Anforderung A2 bezüglich der Ausfallsicherheit wird festgelegt, dass der virtuelle primäre Server i und der virtuelle Standby-Server j nicht auf demselben Substratserver u platziert werden dürfen.

$$\forall i, j \in N^V, i \neq j : \forall u \in N^S : f_{i,j} x_u^i + f_{i,j} x_u^j \leq 1 \tag{2.18}$$

Beispiel 2.9 (Ungleichungen für Anforderung A2)**.** Damit in Abb. 2.4 auch sichergestellt wird, dass der primäre und Standby-Server nicht auf demselben Substratserver platziert werden (siehe Anforderung A1), muss Gleichung 2.18 berücksichtigt werden. Beispielhaft werden hierfür die Ungleichungen für den virtuellen Switch *vsw1* und den Substratswitch *ssw1* vorgestellt.

$$f_{vsw1,vsr1} x_{ssw1}^{vsw1} + f_{vsw1,vsr1} x_{ssw1}^{vsr1} \leq 1$$
$$f_{vsw1,vsr2} x_{ssw1}^{vsw1} + f_{vsw1,vsr2} x_{ssw1}^{vsr2} \leq 1$$

Falls ein virtuelles Netzwerk eine kurze Verzögerungszeit fordert, muss Anforderung A3 eingehalten werden, wodurch ein virtueller Link auf maximal zwei Substratverbindungen platziert werden darf.

$$\forall l_{i,j} \in L^V : \forall p_{u,v} \in P^S : y_{p_{u,v}}^{l_{i,j}} \text{länge}(p_{u,v}) \leq 2 \tag{2.19}$$

2.2.2.4 *Optimierungsziel*

In dieser Arbeit werden als Zielfunktion für das VNE-Problem, analog zu [118], die Minimierung der Kommunikationskosten verwendet. Dazu wird eine Kostenmatrix *kosten* für die Platzierung von virtuellen Links auf Substratpfade benötigt, welche durch den Anwendungsfall bereitgestellt wird.

$$kosten : (P^S \times L^V) \rightarrow \mathbb{R}^+ \tag{2.20}$$

Die in der vorliegenden Arbeit verwendete Zielfunktion ist daher folgende:

$$\text{min:} \sum_{p_{u,v} \in P^S} \sum_{l_{i,j} \in L^V} y_{p_{u,v}}^{l_{i,j}} kosten(l_{i,j}, p_{u,v}) \tag{ILP$_{\text{Ziel}}$}$$

Die Bestimmung der Kostenmatrix kann durch praktische oder theoretische Betrachtungen der virtuellen Netzwerke, des Rechenzentrums oder der wirtschaftlichen Aspekte erfolgen. Für gängige Rechenzentren kann eine Kostenmatrix in [68] gefunden werden, die auch in dieser Arbeit zum Einsatz kommt.

2.2.3 *Lösungsstrategien für das VNE Problem*

Die Forschung zur Lösung des VNE-Problems im Allgemeinen und für Rechenzentren im Speziellen wurde in den vergangenen Jahren intensiv betrieben. So geben die drei Übersichtspapiere [28, 7, 111] einen umfassenden Überblick über die bestehenden Lösungsstrategien aus diesem Bereich und vergleichen bestehende Algorithmen gegeneinander.

Bei der Klassifizierung von VNE-Problemen kann sowohl zwischen (i) offline und online als auch zwischen (ii) statischen und dynamischen Anwendungsfällen unterschieden werden [28]. Bei Offline-Szenarien sind hierbei alle virtuellen Netzwerke und (dynamischen) Änderungen an den Netzwerken und Elementen im Voraus bekannt, wo hingegen bei Online-Szenarien die virtuellen Netzwerke oder Änderungen zeitlich nacheinander abgearbeitet werden müssen ohne weitere Informationen über mögliche zukünftige virtuelle Netzwerke oder Änderungen. Darüber hinaus kann das jeweilige (Offline- oder Online-)Szenario entweder statische oder dynamische Platzierungen unterstützen. Bei statischen Platzierungen wird das Verändern oder Löschen von Netzwerkelementen oder Platzierungen nicht unterstützt, wodurch in diesen Szenarien klassischerweise nur neue virtuelle Netzwerke zusätzlich zu den bestehenden Einbettungen platziert werden. Bei Online-Szenarien entfallen diese Einschränkungen, wodurch zusätzlich das Migrieren und Löschen von Platzierungen mit beachtet werden muss. Obwohl in der Realität meist nur dynamische Online-Szenarien zum Einsatz kommen, wird in der Literatur zur Vereinfachung der Problemstellung oftmals auf statische oder im Funktionsumfang eingeschränkte dynamische Szenarien zurückgegriffen.

Bei der Art der Lösungsstrategie wurde in [28] die Klassifizierung in (i) heuristische, (ii) metaheuristische und (iii) exakte Ansätze getroffen. Dabei reduzieren heuristischen Ansätze (i) den Suchraum des VNE-Problems durch Einschränkung der Netzwerktopologien, Anforderungen oder Zielfunktionen. Somit kann die Laufzeit zum Finden einer annähernd guten Lösung für das VNE-Problem verringert werden. Da diese Algorithmen auf spezielle Anwendungsfälle zugeschnitten sind, ist eine Adaption und Integration von weiteren Anforderungen meist nicht oder nur schwer realisierbar. Zusätzlich können diese Algorithmen keine Garantien über die Einhaltung der Anforderungen oder der Qualität der gefundenen Lösung (z.B. Optimalität) geben. Bei metaheuristischen oder suchbasierten Ansätzen (ii) handelt es sich um problemunabhängige Methoden beispielsweise aus dem Bereich der genetischen Algorithmen [71, 108] oder der Partikelschwarmoptimierung [54, 121]. Diese Methoden besitzen ähnliche Eigenschaften wie heuristische Algorithmen, wodurch wir in dieser Arbeit diese beiden Bereiche nicht weiter getrennt betrachtet werden. Bei den exakten Ansätzen (iii) handelt es sich um mathematische Methoden, die sicherstellen können, dass eine gefundene Lösung korrekt (alle Anforderungen werden eingehalten) und optimal ist, was zulasten der Laufzeit zur Lösung des VNE-Problems geht. Somit sind diese Ansätze nur für kleine Rechenzentren geeignet [110]. Dabei unterstützen sie meist eine Vielzahl von Anwendungsfällen, Anforderungen und Zielfunktionen und können somit leicht adaptiert und integriert werden. Eine etablierte Methode in dieser Domäne ist ILP, da die Ausdrucksmächtigkeit dieser Beschreibungssprache für

Tabelle 2.4: Statische Online-VNE-Ansätze für Rechenzentren

Referenz	Kategorie	Platz	CPU	RAM	HDD	Bb.
[113, 25, 23, 127, 123, 6]	H	✓	×	×	×	✓
[24]	E	✓	×	×	×	✓
[68]	E	✓	×	×	×	×
[114]	E/H	✓	×	×	×	✓
[108]	E/H	×	✓	✓	×	×
[84, 75, 5]	H	×	✓	×	×	✓
[76]	H	×	✓	✓	×	✓
[118]	H	×	✓	✓	✓	×
[17, 16, 119, 79, 41, 120]	H	×	✓	✓	✓	✓

die VNE-Probleme ausreichend ist und hoch entwickelte Solver zur Lösung dieser Problemstellung existieren (z.B. Gurobi oder IBM Cplex).

Der Überblick über Lösungsstrategien für das VNE-Problem für Rechenzentren wird in statische und dynamische Anwendungsfälle unterteilt, wobei zusätzlich noch die unterstützten Ressourcen mit betrachtet werden. Dazu haben wir in der IEEE Explore Digital Library [49] nach den Wörtern *Virtual Network Embedding Data Center* im Mai 2019 gesucht und 33 Veröffentlichungen für diese Problemdomäne gefunden. 21 Veröffentlichungen beziehen sich hierbei auf statische und 12 Veröffentlichungen auf dynamische Anwendungsfälle. Hierbei wurde die Einteilung in exakte (*E*) und heuristische Ansätze (*H*) durchgeführt und als Ressourcen die Prozessorkerne (*CPU*), den Arbeits- (*RAM*), Festplattenspeicher (*HDD*), die Bandbreite (*Bb.*) bzw. ein abstrakter *Platz* betrachtet. Der abstrakte Platz stellt hierbei bei vielen Veröffentlichungen die Größe zur Platzierung eines virtuellen Servers dar, sodass auf einem Platz ein virtueller Server betrieben werden kann. Ein Substratserver kann dabei mehrere Plätze besitzen.

Tabelle 2.4 zeigt die Auswertung der 21 Veröffentlichungen, die statische VNE-Probleme unterstützen. Hierbei können nur neue virtuelle Netzwerke platziert, aber keine virtuellen oder Substratnetzwerke oder deren Ressourcen gelöscht oder geändert werden. Das Symbol ✓ beschreibt hierbei, dass eine Ressource bei der Lösung des VNE-Problems berücksichtigt wird, wo hingegen ein × die Nichtbeachtung dieser Ressource darstellt. Nur bei fünf Veröffentlichungen [24, 68, 114, 108] konnten exakte Ansätze gefunden werden, die eine korrekte (bezüglich der Anforderungen) und optimale Lösung (bezüglich der Zielfunktion) sicherstellen. Allerdings wurde kein exakter Ansatz gefunden, der alle Ressourcen unterstützt. Bei heuristischen Ansätzen konnten 6 Veröffentlichungen [17, 16, 119, 79, 41, 120] gefunden werden, die alle Ressourcen unterstützen.

Für die dynamischen Anwendungsfälle konnten 12 Veröffentlichungen identifiziert werden, wobei hierbei nur heuristische Ansätze zum Einsatz kamen. In Tabelle 2.5 sind die Ergebnisse zusammengefasst, wobei die beiden zusätzlichen Spalten angeben, ob ein Algorithmus das Löschen eines virtuellen Netzwerks (*Lösche VN*) oder das Ändern von Ressourcen (*Änd. Res.*) unterstützt. Dabei konnten wir feststellen, dass nur [61] alle Ressourcen (Prozessorkerne, Arbeits-, Fest-

Tabelle 2.5: Heuristische Algorithmen für dynamische Online-VNE-Probleme in Rechenzentren

Referenz	Platz	CPU	RAM	HDD	Bb.	Lösche VN	Änd. Res.
[124, 66, 122]	✓	×	×	×	✓	✓	×
[107]	✓	×	×	×	✓	×	✓
[112, 22]	✓	×	×	×	✓	✓	✓
[33, 39, 115]	×	✓	×	×	✓	×	✓
[73]	×	✓	✓	×	✓	✓	×
[109]	×	✓	✓	✓	✓	✓	×
[61]	×	✓	✓	✓	✓	✓	✓

plattenspeicher und Bandbreite) und zusätzlich auch die grundlegenden dynamischen Eigenschaften wie Änderung und Löschen von virtuellen Elementen unterstützt. Da es sich hierbei allerdings um einen heuristischen Algorithmus handelt, kann nicht garantiert werden, dass die gefundene Lösung korrekt und optimal ist.

In dieser Arbeit wird im Folgenden ein exakter Ansatz vorgestellt, der sich zur Lösung von statischen und dynamischen Online-VNE-Problemen eignet. Dabei kann durch das Hinzufügen von domänenspezifischer Zusicherungen ein systematischer Übergang von einem exakten Ansatz, der garantiert alle korrekten und optimalen Einbettungen findet, aber in realitätsnahen Szenarien nicht skaliert, zu einem heuristischen Ansatz, der möglicherweise die Korrektheit und Optimalität nicht garantieren kann, aber dafür schneller eine Lösung findet, durchgeführt werden. Somit kann durch die VNE-Problemdefinition die Korrektheit, Optimalität und Geschwindigkeit zur Lösung der Problemstellung bestimmt und auch näher evaluiert werden.

3

MODELLBASIERTE PROBLEMDEFINITION

In diesem Kapitel stellen wir die neu entwickelte modellbasierte Problemdefinition zur Spezifikation von VNE-Problemen vor. Diese modellbasierte Problemdefinition besteht dabei aus einem Metamodell, welches durch ein UML-Klassendiagramm mit OCL-Zusicherungen modelliert wird, und einem Optimierungsziel. In Abschnitt 3.1 führen wir zuerst in die Metamodellierung ein und stellen danach in Abschnitt 3.2 das Klassendiagramm für die modellbasierte Problemdefinition vor. Im Anschluss daran werden weitere Anforderungen durch OCL-Zusicherungen in Abschnitt 3.3 in das Klassendiagramm integriert, sodass alle Anforderungen aus der VNE-Problemdefinition aus Abschnitt 2.2.1 eingehalten werden. Zum Schluss wird in Abschnitt 3.4 noch die Zielfunktion als Optimierungsziel für das VNE-Problem präsentiert.

Somit stellt dieses Kapitel den wissenschaftlichen Beitrag **WB 1** dar, welcher im Artikel [96] präsentiert wurde. Die Einordnung dieses Kapitels in den Gesamtkontext dieser Arbeit ist Abb. 1.1 visualisiert.

3.1 Metamodellierung

Die Metamodellierung ist eine Technik zur abstrakten Beschreibung eines Systems und dessen Zustände als eine Menge von Modellen. Modelle sind hierbei im Allgemeinen als eine abstrakte Abbildung der Realität bzw. eines Systems zu verstehen, um beispielsweise dessen Eigenschaften, Zusammenhänge oder das Verhalten zu beschreiben [58]. Metamodelle dienen hierbei der Modellierung einer Menge von gleichartigen Modellen. Zur Kategorisierung und Vereinheitlichung hat die Object Management Group (OMG)[1] die modellgetriebene Architektur vorgestellt [92, Kapitel 12], wodurch die verschiedenen Abstraktionsebenen von Modellen und deren Beziehungen untereinander modellunabhängig definiert werden können. Dabei werden Modellebenen, unterteilt in Metaebenen, festgelegt. Abbildung 3.1 veranschaulicht anhand des Substratnetzwerks für das Beispiel aus Abb. 2.3a diese Ebenen (M_0, M_1, M_2 und M_3). Zur Abstraktion der Realität stellt die unterste Ebene M_0 die Realität mit deren Objekten als abstrahiertes Domänenmodell dar, wobei nur die relevanten Eigenschaften aus der Realität herausgegriffen und modelliert werden. Auf der nächsten Abstraktionsstufe, der M_1-Ebene, wird die Menge von gültigen M_0-Modellen anhand eines Metamodells beschrieben. Zur Spezifikation eines gültigen M_1-Modells wird das M_2-Modell benötigt, für welches im Allgemeinen eine domänenunabhängige und in vielen

[1] OMG Webseite: https://www.omg.org/ (besucht am 30.01.2020)

Fällen auch objektorientierte Modellierungssprache genutzt wird. Ein etabliertes Beispiel für ein M_2-Modell stellt die Unified Modeling Language (UML) [32] dar. Die höchste und damit auch abstrakteste Ebene ist das M_3-Modell, welches durch die Meta Object Facility (MOF) [92, Abschnitt 12.2.2], eine universelle Modellierungssprache, beschrieben werden kann. Da MOF einen großen Sprachumfang besitzt, wurde Essential Meta Object Facility (EMOF), als eine Untermenge von MOF, spezifiziert, da diese Untermenge für die meisten Modellierungssprachen, wie beispielsweise UML, ausreichend ist.

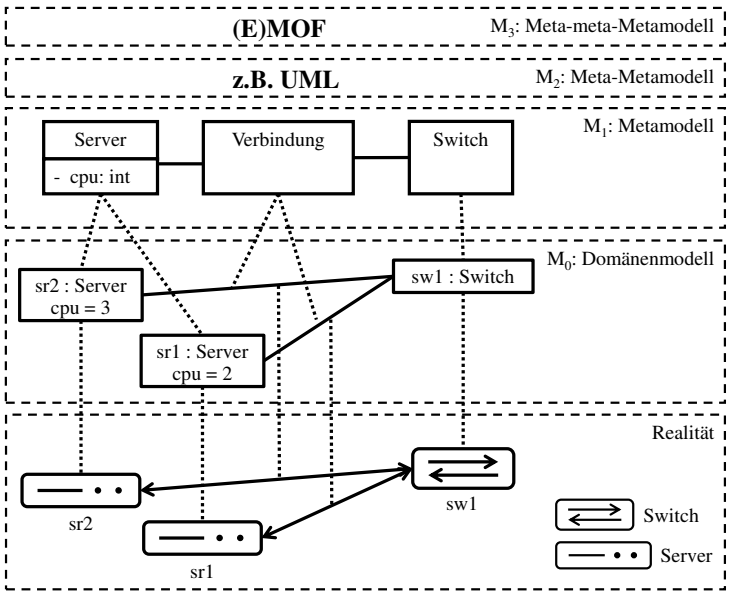


Abbildung 3.1: Modelle nach der OMG Kategorisierung für das Beispiel aus Abb. 2.3a

Beispiel 3.1 (Modelle nach OMG/MOF)**.** In Abb. 3.1 werden anhand eines vereinfachten Beispiels analog zu Abb. 2.3a die verschiedenen Ebenen (M_0, M_1, M_2 und M_3) vorgestellt. Außerhalb dieser Kategorisierung befindet sich die Realität, welche in Abb. 3.1 ganz unten angedeutet ist und aus einem Switch *sw1* mit zwei Servern *sr1* und *sr2* besteht. Die Realität wird nun abstrahiert und in ein Domänenmodell mit Objekten überführt, in welchem nur die relevanten Informationen modelliert werden. In diesem Fall wird für den Switch *sw1* ein Objekt *sw1* vom Typ *Switch* erstellt und für die beiden Server *sr1* und *sr2* jeweils ein Objekt *sr1* und *sr2* vom Typ *Server*. Auch die realen Verbindungen zwischen dem Server und dem Switch werden in das Domänenmodell überführt. Auf der Ebene M_1 ist ein Metamodell vorhanden, welches alle gültigen Domänenmodelle, im vorliegenden Fall alle gültigen Netzwerktopologien, erstellen kann. Dieses Metamodell ist hierbei weiter abstrahiert, sodass keine konkreten Objekte mehr vorhanden sind, sondern nur noch die Klassen *Switch*, *Verbindung* und *Server*, die jeweils eine Menge von Objekten beschreiben. Zur Spezifikation dieses Metamodells kommt ein weiteres Meta-Metamodell aus der Ebene M_2 zum Einsatz. Im vorliegenden Fall ist dieses UML, in welcher Klassen, Attribute oder Assoziationen genutzt

werden können. Die letzte Ebene M_3 stellt MOF bzw. EMOF dar, welches das grundlegende Modell für die Ebene M_2 (UML) zur Verfügung stellt.

Die Metamodellierung wird oft im Kontext der modellbasierten Softwareentwicklung [92] verwendet, da anhand der deklarativ beschriebenen Modelle Spezifikationen, Abhängigkeiten und teilweise auch Funktionalitäten validiert, getestet und automatisiert in Quellcode überführt werden können. Da in dieser Arbeit auch auf Methoden und Techniken aus der modellbasierten Softwareentwicklung zurückgegriffen wird, stellt auch hier die modellbasierte Problemdefinition die Grundlage für die weiteren Methoden und eingesetzten Techniken in dieser Arbeit dar.

3.2 Klassendiagramm

Als grundlegendes Metamodell zur Definition von VNE-Problemen stellen wir ein UML-Klassendiagramm mit OCL-Zusicherungen vor, durch welches virtuelle Netzwerke, Substratnetzwerke und die Platzierungen von virtuellen Netzwerken auf Substratnetzwerken beschrieben wird. Ein Metamodell MM stellt hierbei eine Menge von gültigen Modellinstanzen $\mathfrak{L}(MM)$ dar, die zu diesem Metamodell konform sind [58]. In dieser Arbeit verwenden wir zur Modellierung des Metamodells UML [32] und die Object Constraint Language (OCL) [38], welche jeweils etablierte und standardisierte Modellierungssprachen darstellen. Das entwickelte UML-Klassendiagramm zur Beschreibung von VNE-Problemen ist in Abb. 3.2 abgebildet. Es spezifiziert die virtuellen Netzwerke auf der linken Seite (`Virtuell-Netzwerk`), die Substratnetzwerke auf der rechten Seite (`SubstratNetzwerk`) und die Platzierungen von virtuellen auf Substratelementen als Assoziationen zwischen den `Virtuell-` und `Substrat`-Klassen. Die Präfixe `Virtuell` und `Substrat` zeigen hierbei an, welchem Netzwerk die jeweilige Klasse zugeordnet ist. Anhand der Anforderungen aus Abschnitt 2.2.1 wird nun der Aufbau dieses Klassendiagramms beschrieben und dessen Grenzen aufgezeigt.

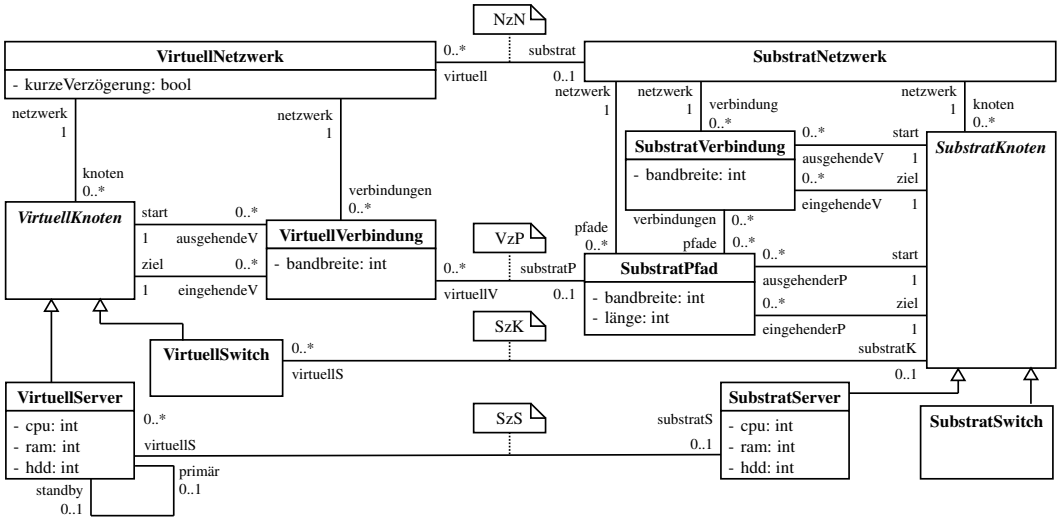


Abbildung 3.2: Klassendiagramm für die modellbasierte Problemdefinition

3.2.1 *Substratnetzwerk*

Jedes Substratnetzwerk (`SubstratNetzwerk`) besteht aus Knoten (`SubstratKnoten`), Verbindungen (`SubstratVerbindung`) oder Pfaden (`SubstratPfad`). Die Knoten repräsentieren hierbei laut Anforderung K1 entweder einen Server (`SubstratServer`) oder einen Switch (`SubstratSwitch`). Ein Substratserver besitzt zusätzlich die Ressourcen Prozessorkerne, Arbeits- und Festplattenspeicher (siehe Anforderung K1) und die Verbindungen und Pfade die Ressource Bandbreite (siehe Anforderung V1). Diese Ressourcen werden als Attribute `cpu`, `ram` und `hdd` für die Klasse `SubstratServer` und `bandbreite` für die beiden Klassen `SubstratVerbindung` und `SubstratPfad` modelliert. Zusätzlich wird noch die Länge eines Substratpfades (siehe Gleichungen 2.2 und 2.5) als Attribut `länge` in der Klasse `SubstratPfad` mit integriert, da sich diese Eigenschaft nur sehr selten ändert und dadurch nicht jedes Mal neu berechnet werden muss. Die Modellierung der Start- und Zielknoten einer Substratverbindung oder eines Substratpfades wird durch die Assoziationen `start` und `ziel` zu den jeweiligen Knoten festgelegt, wobei durch die Multiplizitäten sichergestellt wird, dass jede Verbindung bzw. jeder Pfad genau einen Start- und genau einen Zielknoten hat.

3.2.2 *Virtuelles Netzwerk*

Ein virtuelles Netzwerk (`VirtuellNetzwerk`) ist analog zu einem Substratnetzwerk mit Knoten (`VirtuellKnoten`) und Verbindungen (`VirtuellVerbindung`) aufgebaut. Auch hier können die Knoten entweder vom Typ Server (`VirtuellServer`) oder Switch (`VirtuellSwitch`) sein (siehe Anforderung K1), wobei ein virtueller Server wiederum die Ressourcen Prozessorkerne (`cpu`), Arbeits- (`ram`) und Festplattenspeicher (`hdd`) und eine virtuelle Verbindung die Bandbreite (`bandbreite`) als Ressource besitzt (siehe Anforderung V1). Die virtuelle Verbindung besitzt, wie auch die Substratverbindung, einen Start- (`start`) und einen Zielknoten (`ziel`). Zur Modellierung der Ausfallsicherheit (siehe Anforderung A1) kann jeder virtuelle primäre Server einen Standby-Server besitzen. Dies wird durch die Assoziation `primär` und `standby` repräsentiert, wobei die Multiplizitäten sicherstellen, dass hier jeder primäre maximal einen Standby-Server besitzen kann. Ob ein virtuelles Netzwerk eine kurze Verzögerungszeit (siehe Anforderung A3) fordert, wird durch das Attribut `kurzeVerzögerung` festgelegt. Falls das Attribut den Wert `true` aufweist, muss Anforderung A3 beachtet werden, ansonsten kann diese Anforderung ignoriert werden.

3.2.3 *Platzierungen*

Die Platzierungen von virtuellen Elementen auf Substratelementen wird durch Assoziationen zwischen `Virtuell` und `Substrat`-Elementen modelliert, wobei mehrere virtuelle Elemente (Multiplizität `0..*`) auf maximal ein Substratelement (Multiplizität `0..1`) abgebildet werden können (siehe Anforderungen K2 und V2). Damit wird beispielsweise nicht unterstützt, dass eine virtuelle Verbindung auf zwei parallel genutzte Substratverbindungen abgebildet wird. Eine vorhandene Asso-

Tabelle 3.1: Abkürzungen für Platzierungen

Abkürzung	Platzierung zwischen...
`NzN`	einem virtuellen und einem Substratnetzwerk
`SzS`	einem virtuellen und einem Substratserver
`SzK`	einem virtuellen Switch und einem Substratknoten
`VzP`	einer virtuellen Verbindung und einem Subtratpfad

ziation zwischen zwei Elementen (z.B. einem `VirtuellServer` und einem `SubstratServer`) stellt somit eine Platzierung dar, wobei diese Platzierung nicht notwendigerweise eine gültige (korrekte) Platzierung für dieses VNE-Problem darstellen muss. Im weiteren Verlauf dieser Arbeit werden die Abkürzungen aus Tabelle 3.1 für Platzierungen verwendet.

3.2.4 *Grenzen des Klassendiagramms*

Im Folgenden zeigen wir die Grenzen des Klassendiagramms aus Abb. 3.2 in Bezug auf die Anforderungen aus Abschnitt 2.2.1 auf. Bei der Modellierung der virtuellen und Substratnetzwerke konnten alle Elemente (Server, Switches, Verbindungen und Pfade) modelliert und in Beziehung zueinander gesetzt werden. Somit ist ein Knoten entweder ein Server mit den Ressourcen Prozessorkerne, Arbeits- und Festplattenspeicher oder ein Switch (siehe Anforderung K1). Auch die Anforderung K2, dass ein virtueller Server (Switch) auf genau einem Substratserver (-server oder –switch) platziert werden muss, kann durch `SzS` (`SzK`) im Klassendiagramm abgebildet werden. So stellt die Multiplizität `0..1` bei `substrateP` und `substrateS` sicher, dass maximal ein virtuelles Element, falls eine Platzierung vorliegt, auf dem Substratelement eingebettet wird. Die Überprüfung, dass die Ressourcen der Substratserver nicht überbucht werden (siehe Anforderung K3), lässt sich nicht im Klassendiagramm ausdrücken, da hierbei über eine Menge von Elementen iteriert wird und deren jeweiligen Ressourcen aggregiert werden.

Bei der Modellierung der virtuellen und Substratverbindungen wurde die Ressource Bandbreite mitberücksichtigt (siehe Anforderung V1). Allerdings kann nicht garantiert werden, dass der Start- bzw. Zielknoten einer virtuellen Verbindung auf den Start- bzw. Zielknoten des Substratpfades platziert wird (siehe Anforderung V2). Im Klassendiagramm kann lediglich sichergestellt werden, dass virtuelle Knoten auf Substratknoten platziert werden (siehe Anforderung K2). Bei der Anforderung V3, der Sicherstellung, dass die Ressourcen der Substratverbindungen bzw. –pfade nicht überbucht werden, verhält es sich analog zu Anforderung K3, wodurch diese nicht im Klassendiagramm abgebildet werden kann.

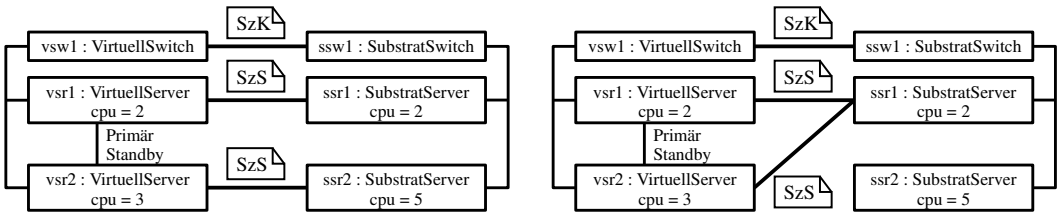
Bei den anwendungsfallspezifischen Anforderungen, konnte Anforderung A1, dass jeder virtuelle primäre Server maximal einen Standby-Server besitzen darf, modelliert werden. Allerdings kann nicht sichergestellt werden, dass der primäre und der Standby-Server nicht der gleiche Server sind. Anforderung A2, dass diese beiden Server nicht auf demselben Substratserver platziert werden, kann nicht

Tabelle 3.2: Umgesetzte Anforderungen aus Abschnitt 2.2.1 im Klassendiagramm aus Abb. 3.2

Anforderung		**Erfüllt**
Knotenspezifische Anforderung	K1	✓
	K2	✓
	K3	×
Verbindungsspezifische Anforderung	V1	✓
	V2	○
	V3	×
Anwendungsfallspezifische Anforderung	A1	○
	A2	×
	A3	×

modelliert werden. Auch Anforderung A3, die Realisierung einer kurzen Verzögerungszeit, lässt sich in dieser Form nicht im Klassendiagramm abbilden.

Eine Zusammenfassung der im Klassendiagramm umgesetzten Anforderungen wird in Tabelle 3.2 dargestellt, wobei ein ✓ anzeigt, dass eine Anforderung komplett, ein ○, dass sie teilweise, und ein ×, dass sie gar nicht modelliert werden konnte. Daher wird neben dem Klassendiagramm noch eine Methodik bzw. Beschreibungssprache benötigt, um die fehlenden Anforderungen sicherzustellen.



(a) Korrekte Lösung **(b)** Unzulässige Lösung

Abbildung 3.3: Konkrete Instanzen des Klassendiagramms aus Abb. 3.2 mit möglichen Lösungen für das VNE-Problem

Beispiel 3.2 (Instanzen des Klassendiagramms)**.** In Abb. 3.3 sind zwei konkrete Instanzen des Klassendiagramms aus Abb. 3.2 dargestellt. Bei diesen Instanzen stellen die durchgängigen Verbindungslinien zwischen den virtuellen Elementen und den Substratelementen Platzierungen dar, die konform zum Klassendiagramm sind. Hierbei repräsentiert in Abb. 3.3a die Assoziation von *vsw1* →*ssw1* eine `SzK` und die Assoziationen *vsr1* →*ssr1* und *vsr2* →*ssr2* eine `SzS` Platzierung. Da im Klassendiagramm nicht alle Anforderungen sichergestellt werden können (siehe Tabelle 3.2), kann eine Instanz des Klassendiagramms eine gültige (korrekte) Lösung (siehe Abb. 3.3a) oder eine unzulässige Lösung (siehe Abb. 3.3b) für das VNE-Problem darstellen. Die Lösung in Abb. 3.3b ist unzulässig, da sowohl die Anforderung K3 als auch die Anforderung A2 verletzt wird. Bei Anforderung K3 wird gefordert, dass

die Summe der virtuellen *cpu* Ressourcen kleiner gleich der Substratressourcen sein muss ($vsr1.cpu + vsr2.cpu \leq ssr1.cpu$). Anforderung A2 legt fest, dass der primäre Server *vsr1* und der Standby-Server *vsr2* nicht auf demselben Substratserver platziert werden dürfen.

3.3 OCL-Zusicherungen

In Abschnitt 3.2 haben wir ein Klassendiagramm modelliert, in welches allerdings nicht alle Anforderungen aus Abschnitt 2.2.1 integriert werden konnten, sodass korrekte Lösungen für das VNE-Problem nicht alleine durch das Klassendiagramm sichergestellt werden können. Somit nutzen wir OCL [104, 38], ein Bestandteil von UML, zur formalen Modellierung von Zusicherungen für Klassendiagramme (z.B. Invarianten, Voraussetzungen, Nachbedingungen, Anfangs- und abgeleitete Werte, Definitionen und Wächter). Dadurch können beispielsweise Wertebereiche von Attributen eingeschränkt oder Beschränkungen zwischen Objekten realisiert werden. Da zur Definition für etablierte VNE-Probleme nicht der volle Umfang von OCL benötigt wird, beschränken wir uns in dieser Arbeit auf eine Untermenge von OCL, die alle Anforderungen aus Abschnitt 2.2.1 erfüllen kann. Diese Teilmenge besteht aus der Essentail Object Constraint Language (EOCL) [38, 80], die auf der Prädikatenlogik erster Ordnung, sowie arithmetischen und Mengen-Operatoren aufbaut und der Operation `iterate()` [38], die jedoch auf die Aggregation einzelner Elemente der zu iterierenden Menge beschränkt ist (analog zur Operation `sum()`). Da die Operation `iterate()` für die Aufsummierung jeder Ressource (Prozessorkerne, Arbeits-, Festplattenspeicher und Bandbreite) erforderlich ist, definieren wir für jede Ressource eine eigene zusätzliche Hilfsfunktion. Als Beispiel wird im Folgenden die Hilfsfunktion für die Summierung der Ressourcen für die Prozessorkerne über eine Menge von Elementen (z.B. virtuelle Server) vorgestellt. Die anderen Hilfsfunktionen für Arbeits-, Festplattenspeicher und Bandbreite werden analog dazu definiert.

$$\begin{aligned}&\textbf{def}\text{: cpuSum (col : Collection(T)) : Integer}\\&\qquad\qquad = \text{col.iterate(elem : T; sum : Integer = 0} \mid \text{elem.cpu + sum)}\end{aligned} \tag{3.1}$$

Im weiteren Verlauf der Arbeit unterscheiden wir zwischen den Sprachen OCL, EOCL und COCL, wobei EOCL (Prädikatenlogik erster Ordnung, sowie arithmetischen und Mengen-Operatoren) und COCL eine disjunktive Untermenge von OCL darstellen. Analog hierzu repräsentieren $\mathfrak{L}(OCL)$, $\mathfrak{L}(EOCL)$ und $\mathfrak{L}(COCL)$ Mengen von Zusicherungen aus der jeweiligen Sprache (OCL, EOCL oder COCL), wobei folgendes gilt:

$$\mathfrak{L}(OCL) = \mathfrak{L}(EOCL) \cup \mathfrak{L}(COCL) \tag{3.2}$$

Eine konkrete Menge an Zusicherungen wird durch C_{Suffix} repräsentiert, welche eine Untermenge von $\mathfrak{L}(OCL)$ ($C_{\text{Suffix}} \subseteq \mathfrak{L}(OCL)$) darstellt. Somit definieren wir die folgenden Mengen von Zusicherungen:

$$\begin{aligned} C_{\text{OCL}} &\subseteq \mathfrak{L}(OCL) \\ C_{\text{EOCL}} &= C_{\text{OCL}} \cap \mathfrak{L}(EOCL) \\ C_{\text{COCL}} &= C_{\text{OCL}} \cap \mathfrak{L}(COCL) = \mathfrak{L}(OCL) \setminus \mathfrak{L}(EOCL) \end{aligned} \tag{3.3}$$

In diesem Kontext stellen $\mathfrak{L}(C_{\text{OCL}})$, $\mathfrak{L}(C_{\text{EOCL}})$ und $\mathfrak{L}(C_{\text{COCL}})$ Mengen von Modellen dar, welche die konkrete Menge an Zusicherungen C_{OCL}, C_{EOCL} bzw. C_{COCL} einhalten. Es gilt:

$$\mathfrak{L}(C_{\text{OCL}}) = \mathfrak{L}(C_{\text{EOCL}}) \cap \mathfrak{L}(C_{\text{COCL}}) \tag{3.4}$$

Ausgehend von Tabelle 3.2 werden nun die Anforderungen die nicht durch das Klassendiagramm sichergestellt werden, als OCL-Zusicherungen formuliert und mit in das Klassendiagramm integriert. Dadurch kann durch das komplette Metamodell (Klassendiagramm und OCL-Zusicherungen) garantiert werden, dass jede Modellinstanz, die konform zu diesem Metamodell ist, eine korrekte Lösung für das VNE-Problem darstellt, wobei diese Lösungen nicht notwendigerweise auch optimal sein müssen.

3.3.1 *Knotenspezifische Anforderungen*

Bei den knotenspezifischen Anforderungen konnten nur die beiden Anforderungen K1 und K2 durch das Klassendiagramm eingehalten werden (siehe Tabelle 3.2). Zur Sicherstellung von Anforderung K3, welche fordert, dass die Substratressourcen nicht durch die virtuellen Ressourcen überbucht werden, dürfen die aggregierten virtuellen Ressourcen (Prozessorkerne, Arbeits- und Festplattenspeicher) aller virtuellen Server, die auf einem Substratserver platziert werden, die Ressourcen dieses Substratservers nicht überschreiten. Die Integration der folgenden drei OCL-Zusicherungen garantiert dies, wobei die Hilfsfunktionen aus Gleichung 3.1 verwendet werden.

context SubstratServer **inv** cpuSum(self.virtuellS) $\leq$ self.cpu (C_{cpu})
context SubstratServer **inv** ramSum(self.virtuellS) $\leq$ self.ram (C_{ram})
context SubstratServer **inv** hddSum(self.virtuellS) $\leq$ self.hdd (C_{hdd})

Beispiel 3.3 (OCL-Zusicherung C_{cpu})**.** Die OCL-Zusicherung C_{cpu} sieht für die konkrete Instanz aus Abb. 3.3a wie folgt aus:

$$\begin{aligned} \mathit{vsr1.cpu} \leq \mathit{ssr1.cpu} &\mathrel{\widehat{=}} 2 \leq 2 \\ \mathit{vsr1.cpu} \leq \mathit{ssr2.cpu} &\mathrel{\widehat{=}} 3 \leq 5 \end{aligned}$$

Somit kann diese Instanz unter der Bedingung, dass auch alle anderen Zusicherungen erfüllt sind, eine korrekte Lösung für das VNE-Problem darstellen. Bei Abb. 3.3b hingegen wird die OCL-Zusicherung C_{cpu}

$$vsr1.cpu + vsr2.cpu \leq ssr1.cpu \mathrel{\widehat{=}} 5 \leq 2$$

verletzt, wodurch diese Instanz keine korrekte Lösung für das VNE-Problem darstellt.

3.3.2 *Verbindungsspezifische Anforderungen*

Bei den verbindungsspezifischen Anforderungen (siehe Tabelle 3.2) konnte Anforderung V1 komplett, Anforderung V2 teilweise und Anforderung V3 nicht durch das Klassendiagramm sichergestellt werden. Anforderung V2 garantiert hierbei, dass eine virtuelle Verbindung genau auf einem Substratpfad platziert wird, wobei die jeweiligen Start- und Zielknoten der virtuellen Verbindung auf den Start- und Zielknoten des Substratpfades platziert sein müssen. Da im Klassendiagramm lediglich sichergestellt wird, dass ein virtueller Knoten auf einem Substratknoten platziert wird, müssen nun OCL-Zusicherungen garantieren, dass die jeweiligen Start- bzw. Zielknoten des virtuellen und des Substratknotens aufeinander platziert sind. Dabei muss natürlich Anforderung K2 mit beachtet werden, da ein virtueller Server (Switch) nur auf einem Substratserver (-server oder –switch) platziert werden darf. Dies resultiert in den folgenden beiden OCL-Zusicherungen:

context VirtuellVerbindung **inv**
  if self.start.oclIsTypeOf(VirtuellServer)
    then self.start.oclAsType(VirtuellServer).substratS = self.substratP.start
    else self.start.oclAsType(VirtuellSwitch).substratK = self.substratP.start
  endif

(C_{start})

context VirtuellVerbindung **inv**
  if self.ziel.oclIsTypeOf(VirtuellServer)
    then self.ziel.oclAsType(VirtuellServer).substratS = self.substratP.ziel (C_{ziel})
    else self.ziel.oclAsType(VirtuellSwitch).substratK = self.substratP.ziel
  endif

Die OCL-Zusicherung für Anforderung V3, analog zur Anforderung K3, stellt sicher, dass die Bandbreite aller virtuellen Verbindungen die auf einem Substratpfad platziert werden, die Bandbreite dieses Substratpfades nicht übersteigt. Somit ergibt sich die folgende OCL-Zusicherung:

context SubstratVerbindung **inv**
  self.bandbreiteSum(self.pfade.virtualL) $\leq$ self.bandbreite

(C_{bb})

3.3.3 *Anwendungsspezifische Anforderungen*

Bei den anwendungsfallspezifischen Anforderungen konnte Anforderung A1 teilweise durch das Klassendiagramm sichergestellt werden. Hierbei konnte nicht garantiert werden, dass der primäre und der Standby-Server unterschiedliche Server sind, was durch die folgende OCL-Zusicherung nun respektiert wird:

$$\textbf{context}\ \text{VirtuellServer}\ \textbf{inv}\ \text{self.standby}\ <>\ \text{self} \qquad (C_{a1})$$

Sowohl Anforderung A2 als auch Anforderung A3 müssen durch weitere OCL-Zusicherungen modelliert werden. Bei Anforderung A2 wird gefordert, dass ein virtueller primärer und virtueller Standby-Server nicht auf demselben Substratserver platziert werden dürfen. Somit ergibt sich für diese Anforderung die folgende OCL-Zusicherung:

$$\textbf{context}\ \text{VirtuellServer}\ \textbf{inv}\ \text{self.standby.substratS}\ <>\ \text{self.substratS} \qquad (C_{a2})$$

Beispiel 3.4 (OCL-Zusicherung C_{a2})**.** Die OCL-Zusicherung C_{a2} für die konkrete Instanz aus Abb. 3.3a

$$vsr1.standby.substratS <> vsr1.substratS \mathrel{\widehat{=}} ssr2 <> ssr1$$

wird eingehalten, wodurch dies eine korrekte Lösung für das VNE-Problem unter der Bedingung, dass alle anderen Anforderungen auch erfüllt sind, darstellen kann. Bei Abb. 3.3b ergibt sich folgende konkrete Zusicherung

$$vsr1.standby.substratS <> vsr1.substratS \mathrel{\widehat{=}} ssr1 <> ssr1$$

welche nicht erfüllt wird, wodurch Abb. 3.3b keine korrekte Lösung für das VNE-Problem darstellt.

Für die Anforderung A3 muss garantiert werden, dass, wenn das virtuelle Netzwerk eine kurze Verzögerungszeit fordert (`kurzeVerzögerung = true`), die virtuellen Verbindungen nur auf Substratpfade der Länge kleiner oder gleich 2 platziert werden. Dies wird durch die folgende OCL-Zusicherung erreicht:

$$\textbf{context}\ \text{VirtuellVerbindung}\ \textbf{inv}\ \text{self.substratP.länge}\ \leq\ 2 \qquad (C_{a3})$$

3.4 Optimierungsziel

Zur Spezifikation des Optimierungsziels für das VNE-Problem nutzen wir eine Erweiterung von OCL zur Modellierung von Zielfunktionen für Optimierungsprobleme. Diese Erweiterung [57] erlaubt es, Zielfunktionen in OCL-Ausdrücke zu integrieren und diese Zielfunktionen zu minimieren bzw. zu maximieren. Als Optimierungsziel wird analog zur ILP-basierten Problembeschreibung (siehe Abschnitt 2.2.2.2), die Minimierung der aggregierten Kommunikationskosten aller virtuellen Verbindungen verwendet (siehe Gleichung ILP_{Ziel}). Hierbei beschreibt

die zur Laufzeit konstante Kostenmatrix *kosten(l,p)*, die Kommunikationskosten für jede mögliche Platzierung einer virtuellen Verbindung l auf einen Substratpfad p. Für gängige Rechenzentren kann eine Kostenmatrix in [68] gefunden werden. Zur Veranschaulichung dient hierbei die Kostenfunktion für ein 2-Tier-Netzwerk mit einer VL2-Topologie von [68], welche wie folgt definiert ist:

def: kosten(l : VirtuellVerbindung, p : SubstratPfad) : Integer

$$= \begin{cases} 0 & \text{wenn } p.hops = 0, \\ l.bandbreite & \text{wenn } p.hops = 1, \\ 5 \cdot l.bandbreite & \text{wenn } p.hops > 1 \end{cases} \tag{3.5}$$

Zur Berechnung der Gesamtkommunikationskosten müssen die Kosten für alle möglichen Platzierungen von virtuellen Verbindungen auf Substratpfade aufsummiert werden. Dabei muss zusätzlich sichergestellt werden, dass alle virtuellen Verbindungen auf Substratpfade und alle virtuellen Server (Switches) auf Substratserver (-server oder -switches) abgebildet sind. Somit ergibt sich die folgende Zielfunktion:

```
context VirtuellNetzwerk
    minimize: self.verbindungen →
      iterate(l; total : Integer = 0 | total + kosten(l,l.substrateP))
    inv: self.verbindungen → exists(substratP)
    inv: self.knoten → forAll(k |                                        (3.6)
      if k.oclIsTypeOf(VirtuellServer)
        then k.asTypeOf(VirtuellServer).substratS → notEmpty()
        else k.asTypeOf(VirtuellSwitch).substratK → notEmpty()
      endif)
```

4

MODELLBASIERTE ENTWICKLUNGSMETHODIK

In diesem Kapitel präsentieren wir das neuartige Konzept eines modellbasierten Ansatzes zur Lösung von VNE-Problemen (engl. model-driven virtual network embedding, kurz MdVNE) und einer per-Konstruktion-korrekten Methodik zur Erstellung einer Konfiguration für MdVNE. MdVNE basiert hierbei auf der zuvor vorgestellten modellbasierten Problemdefinition und nutzt Techniken aus der Modelltransformation und der ganzzahligen linearen Optimierung, um das Optimierungsproblem für die Einbettung von virtuellen Netzwerken in ein Substratnetzwerk zu lösen. Mithilfe der per-Konstruktion-korrekten Methodik und der modellbasierten Problemdefinition kann hierbei eine MdVNE-Konfiguration erstellt werden, die korrekte und optimale Lösungen garantiert. Wir beschränken uns in diesem Kapitel zunächst auf statische Online-Szenarien, um die verwendeten Methoden, Techniken und Vorgehensweisen in einem vereinfachten Anwendungsszenario zu erläutern.

Zu Beginn stellen wir in Abschnitt 4.1 zunächst den modellbasierten Ansatz MdVNE vor und führen in Abschnitt 4.2 in die Grundlagen der Tripel-Graph-Grammatiken ein, die zur Modelltransformation in diesem Kapitel verwendet wird. Danach präsentieren wir in Abschnitt 4.3 die per-Konstruktion-korrekte Methodik, wobei wir zuerst auf die Erstellung eines Regelsatzes für die Modelltransformationen eingehen (Abschnitt 4.3.1) und danach die Ableitung einer mathematischen Formulierung für die ganzzahlige lineare Optimierung zur Lösung des VNE-Problems vorstellen (Abschnitt 4.3.2). Zum Schluss beweisen wir in Abschnitt 4.3.3, dass die vorgestellte Konstruktionsmethodik korrekte, in Bezug auf die Anforderungen, und optimale Lösungen, in Bezug auf das Optimierungsziel, findet. In Abb. 4.1 sind die einzelnen Schritte und eine Übersicht über das Kapitel abgebildet.

Somit stellt dieses Kapitel die beiden wissenschaftlichen Beitrage **WB 2** und **WB 3** dar, welche als Vorversionen in [97, 98, 96] präsentiert wurden. Eine schematische Darstellung zum Aufbau des Kapitels und dessen Einordnung ist in Abb. 4.1 zu finden.

4.1 Einführung in MdVNE

Zur Lösung von statischen VNE-Problemen stellen wir in diesem Abschnitt den MdVNE-Ansatz vor, mit welchem sich sowohl heuristische als auch korrekte und optimale VNE-Algorithmen umsetzen lassen. Hierzu werden MT-Technologien verwendet, um den Suchraum für ein gegebenes VNE-Problem zu reduzieren, und ILP-Technologien, um innerhalb dieses reduzierten Suchraums eine korrekte,

in Bezug auf die Anforderungen, und optimale Lösung, in Bezug auf das Optimierungsziel, zu erhalten. Eine schematische Darstellung von MdVNE ist in Abb. 4.1 zu finden. Im Folgenden werden wir uns auf korrekte und optimale VNE-Algorithmen konzentrieren, da diese durch den enormen Suchraum und der Menge an möglichen Platzierungen eine besondere Herausforderung darstellen.

Bevor MdVNE angewendet werden kann, wird eine *MdVNE-Konfiguration* erstellt, die aus einer *Modelltransformation (MT)*- und einer *ILP-Konfiguration* besteht, wobei die MT-Konfiguration einen MT-Regelsatz für die Kandidatengenerierung und die ILP-Konfiguration eine ILP-Formulierung für die Kandidatenselektion enthält. Somit ist in diesen Konfigurationen Domänenwissen (z.B. über zulässige Platzierungen, Anforderungen oder dem Optimierungsziel) integriert, welche unabhängig von der konkreten Problemstellung sind (tatsächliche Abfolge von virtuellen Netzwerken oder der konkreten Instanz des Substratnetzwerks). Anhand der MT- und der ILP-Konfiguration wird nun ausführbarer Quellcode erzeugt, welcher zur Lösung von konkreten Instanzen eines VNE-Problems verwendet werden kann.

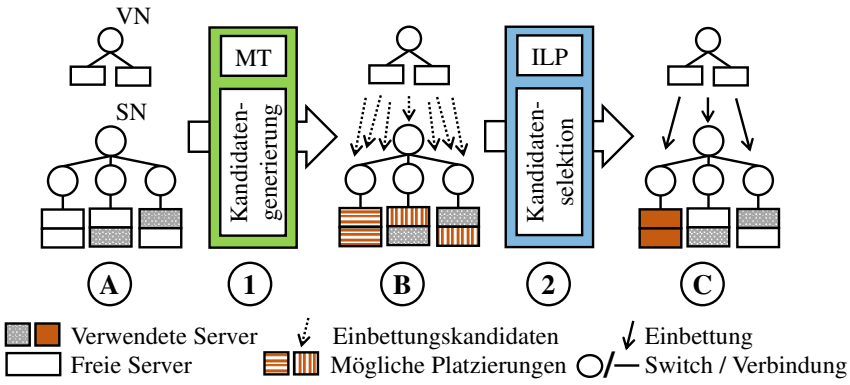


Abbildung 4.1: Schematische Darstellung von MdVNE

Zur Laufzeit werden ein oder mehrere virtuelle Netzwerke (VN), die konkrete Instanzen von virtuellen Netzwerken repräsentieren, erstellt, welche in das Substratnetzwerk (SN) eingebettet werden sollen (Abb. 4.1 (A)). Dabei sind im Allgemeinen im Substratnetzwerk bereits eingebettete virtuelle Netzwerke (repräsentiert durch graue Rechtecke) und frei verfügbare Ressourcen (repräsentiert durch weiße Rechtecke) vorhanden.

Die Einbettung der virtuellen Netzwerke erfolgt nun iterativ, wobei in jeder Iteration ein oder mehrere noch nicht bearbeitete virtuelle Netzwerke ausgewählt und eingebettet werden. Die Einbettung erfolgt anhand der beiden Schritte: Kandidatengenerierung (1) und Kandidatenselektion (2). Die Generierung aller Platzierungskandidaten erfolgt hierbei anhand des MT-Regelsatzes, welcher Kandidaten herausfiltert, die knoten-, verbindungs- oder anwendungsspezifische Anforderungen verletzen, welche durch den MT-Regelsatz ausgedrückt werden können. Dabei muss der MT-Regelsatz sicherstellen, dass in der Menge von Platzierungskandidaten mindestens eine korrekte und optimale Platzierung enthalten ist. Hierbei wird der Kompromiss deutlich, dass einerseits korrekte und optimale Platzierungskandidaten nicht verworfen werden dürfen, aber andererseits die Menge an Platzierungskandidaten so weit wie möglich reduziert werden soll. So-

mit führt der Schritt der Kandidatengenerierung zu einer Menge von möglichen Platzierungskandidaten, welche in Abb. 4.1 Ⓑ durch die gepunkteten Pfeile und die schraffierten Platzierungen von Substratservern angedeutet ist. Diese Menge kann jedoch Platzierungskandidaten enthalten, die (i) unzulässig sind (d.h. das Metamodell oder eine OCL-Zusicherung verletzen, welche nicht durch den MT-Regelsatz ausgedrückt werden kann) oder (ii) korrekt sind, aber in Bezug auf das Optimierungsziel eine suboptimale Lösung darstellen. Im Folgenden verwenden wir zur Spezifikation des MT-Regelsatzes für die MT-Konfiguration von MdVNE eine Tripel-Graph-Grammatik (TGG). Eine TGG [89] beschreibt hierbei mögliche Abbildungen zwischen zwei Graphen (im vorliegenden Fall zwischen einem virtuellen und einem Substratnetzwerk) durch einen dritten Korrespondenzgraphen (hier die Platzierungen eines virtuellen Netzwerks in einem Substratnetzwerk) in deklarativer und regelbasierter Weise. Die erstellten TGG-Regeln können auf vielfältige Weise operationalisiert werden. Im MdVNE-Ansatz sind die beteiligten Graphen das virtuelle und das Substratnetzwerk, wobei der Korrespondenzgraph zu Beginn fehlt und erst in Schritt ① erstellt wird. Diese Art der Operationalisierung wird auch als Konsistenzprüfung bezeichnet [63]. Zur Automatisierung der Quellcodegenerierung auf Basis der TGG-Regeln wird für MdVNE das MT-Tool eMoflon [62] verwendet.

Danach wird in der Kandidatenselektion (Schritt ② aus Abb. 4.1) eine korrekte und optimale Platzierung aus der Menge der Platzierungskandidaten aus Schritt ① ausgewählt. Die Auswahl erfolgt anhand der ILP-Formulierungen und dem Optimierungsziel aus der ILP-Konfiguration und den ILP-Formulierungen, die im Schritt ① der Kandidatengenerierung erstellt wurden, um die Konsistenz zur VNE-Problemdefinition zu gewährleisten. Die Lösung der resultierenden ILP-Formulierung wird durch einen ILP-Löser bestimmt, wodurch die gewonnene Lösung bzw. Platzierung korrekt in Bezug auf die Anforderungen und optimal in Bezug auf das Optimierungsziel ist. Der letzte Schritt Ⓒ aus Abb. 4.1 platziert die durch den ILP-Löser berechnete Lösung im Substratnetzwerk (repräsentiert durch die durchgehenden Pfeile und die braun ausgefüllten Substratserver). Danach kann das nächste virtuelle Netzwerke eingebettet werden.

Eine zentrale Herausforderung beim Einsatz von MdVNE stellt die Erstellung der MT- und ILP-Konfiguration dar, da hierfür tiefgreifende Kenntnisse aus den Bereichen MT, ILP und der VNE-Domäne notwendig sind. Außerdem ist dieser Schritt fehleranfällig und zeitaufwändig; ein Nachweis, dass die erstellte Konfiguration garantiert korrekte und optimale Lösungen für die geforderten Anforderungen liefert, ist meist nur mit viel Aufwand durchführbar. Daher wird im Folgenden eine systematische Konstruktionsmethodik vorgestellt, die per Konstruktion eine korrekte und optimale MT- und ILP-Konfiguration für MdVNE aus einer deklarativen modellbasierten Spezifikation ableitet, welche in Kapitel 3 vorgestellt wurde. Diese Konstruktionsmethodik stellt zusätzlich sicher, dass mindestens eine korrekte und optimale Lösung für das VNE-Problem gefunden wird, falls eine solche Lösung existiert.

4.2 Grundlagen von Tripel-Graph-Grammatiken

Wir beginnen mit einer kurzen Einführung in TGGs [89] anhand der TGG-Regeln aus Abb. 4.3b und 4.8a. Eine *TGG* besteht aus einem TGG-Regelsatz und wird im Kontext eines linken und rechten Metamodells (kurz linke und rechte Seite) und eines korrespondierenden Metamodells (kurz Korrespondenzmodell) definiert. In dieser Arbeit repräsentiert die linke Seite das Metamodell, welches die virtuellen Netzwerke beschreibt (Klassen mit dem Präfix `Virtuell` aus Abb. 3.2), die rechte Seite das Metamodell für das Substratnetzwerk (Klassen mit dem Präfix `Substrat` aus Abb. 3.2) und das Korrespondenzmodell die Einbettungen bzw. Platzierungskandidaten zwischen virtuellen und Substratnetzwerk-Elementen (Assoziationen markiert mit Kommentaren in der Mitte von Abb. 3.2). Eine Korrespondenzklasse im Korrespondenzmodell hat immer genau zwei Assoziationen zu einer Klasse der linken und rechten Seite. Die Assoziation zur linken Seite entspricht den `virtuell*` Assoziationen aus dem Metamodell der modellbasierten Problemdefinition und die Assoziation zur rechten Seite der `substrat*` Assoziation.

Eine *TGG-Regel* besteht hierbei aus Objektvariablen (z.B. `SubstratNetzwerk` in Abb. 4.3b), Verbindungsvariablen (z.B. `knoten` zwischen einen `SubstratNetzwerk` und einem `SubstratSwitch` in Abb. 4.3b) und relationalen Attributeinschränkungen (z.B. `vsr.cpu` $\leq$ `ssr.cpu` in Abb. 4.8a). Dabei beschreiben die Objekt- und Verbindungsvariablen Platzhalter für Objekte und Verbindungen aus einem der drei Modelle der TGG. Hierbei kann eine Variable entweder den notwendigen Kontext, der für die Anwendbarkeit der TGG-Regel erforderlich ist (schwarz dargestellt), oder ein Modellelement, welches durch die Anwendung dieser TGG-Regel erstellt wird (grün dargestellt und mit einem ++ beschriftet), beschreiben. Eine relationale Attributeinschränkung vergleicht die Werte der Attribute von (zwei) Objektvariablen mit binären Operatoren (z.B. < oder =).

> **Beispiel 4.1** (Variablen in TGG-Regeln)**.** Die TGG-Regel in Abb. 4.3b kann nur angewendet werden, wenn ein `SubstratNetzwerk` existiert. Nach Ausführung dieser Regel wird diesem `SubstratNetzwerk` ein `SubstratSwitch` hinzugefügt und eine Verbindung zwischen dem `SubstratNetzwerk` und dem `SubstratSwitch` erstellt.

Eine TGG beschreibt hierbei eine Grammatik, welche in dieser Arbeit die Menge aller erlaubten virtuellen Netzwerken, Substratnetzwerken und den Einbettungen von virtuellen Elementen in Substratelemente repräsentiert. Dabei liegt der Fokus bei der *Operationalisierung* einer TGG in dieser Arbeit auf der Herstellung der Konsistenz zwischen der linken und der rechten Seite (*Konsistenzprüfung*). Das bedeutet, dass die linken und rechten Modellinstanzen schon vorhanden sind und nur das Korrespondenzmodell bzw. die Korrespondenzmodellinstanz (d.h. die `virtuell*-substrat*` Assoziationen aus Abb. 3.2) fehlt und hinzugefügt werden soll [63]. In dieser Arbeit gehen wir bei den MT- bzw. TGG-Regeln davon aus, dass unterschiedliche Variablen auch auf unterschiedliche Modellelemente abgebildet werden (*injektive Mustererkennung* [47]).

Zur Erstellung der linken und rechten Modellinstanzen sowie des Korrespondenzmodells wird folgender Ablauf durchgeführt:

1. Das Substratnetzwerk wird beispielsweise durch einen MT-Regelsatz erstellt.

2. Zur Konsistenzprüfung des Substratnetzwerks werden die entsprechenden TGG-Regeln verwendet, wodurch geprüft werden kann, ob das Substratnetzwerk strukturell korrekt aufgebaut ist.

3. Die virtuellen Netzwerke werden analog zum Substratnetzwerk beispielsweise durch einen MT-Regelsatz erstellt.

4. Zur Konsistenzprüfung der virtuellen Netzwerke werden nun die entsprechenden TGG-Regeln für die virtuellen Netzwerke ausgeführt. Dadurch kann simultan die strukturelle Korrektheit der virtuellen Netzwerke geprüft und die Menge der Platzierungskandidaten aus dem Korrespondenzmodell generiert werden.

Beispiel 4.2 (Erzeugung der Modellinstanzen und Konsistenzprüfungen)**.** Zuerst wird das Substratnetzwerk aus Abb. 3.3a erstellt (Schritt 1). Danach wird mithilfe der entsprechenden TGG-Regeln (siehe Abb. 4.3) eine Konsistenzprüfung durchgeführt, um sicherzustellen, dass das Substratnetzwerk strukturell korrekt aufbaut ist (Schritt 2). Da die TGG-Regel aus Abb. 4.3a keinen weiteren Kontext benötigt, wird diese Regel zuerst ausgeführt und stellt somit die Axiomregel [3] dar.

Nachdem das Substratnetzwerk aufgebaut und überprüft wurde, können wir die virtuellen Netzwerke erzeugen (Schritt 3). Nun wird simultan die korrekte Struktur der virtuellen Netzwerke überprüft und das Korrespondenzmodell aufgebaut (Schritt 4). Dazu werden die TGG-Regeln aus Abb. 4.4 verwendet, wobei das Substratnetzwerk (rechte Seite) als Kontext dient. Nachdem das Korrespondenzmodell erzeugt wurde, können die Korresponzlinks in Platzierungskandidaten (z.B. `SzS` aus Abb. 3.3a) überführt werden.

4.3 Per-Konstruktion-korrekte Methodik

In diesem Abschnitt stellen wir die per-Konstruktion-korrekte Methodik für den MdVNE-Ansatz dar, welche es ermöglicht anhand einer modellbasierten Problemdefinition systematisch eine MdVNE-Konfiguration (MT- und ILP-Konfiguration) für den MdVNE-Ansatz zu erstellen. Weiter wird durch diese systematische Konstruktionsmethodik sichergestellt, dass die Menge an Lösungen, die MdVNE durch die entstandene Konfiguration berechnet, korrekte und optimale Lösungen für das VNE-Problem liefert. Eine schematische Übersicht über die Konstruktionsmethodik und die Integration in MdVNE ist in Abb. 4.2 zu finden, welche auch die Schritte zur Erstellung der MdVNE-Konfiguration und die Struktur der folgenden Abschnitte wiedergibt.

Ausgangspunkt für diese per-Konstruktion-korrekte Methodik bildet die modellbasierte Problemdefinition aus Kapitel 3, die aus den drei Teilen Klassendiagramm, OCL-Zusicherungen und Zielfunktion besteht. Das Klassendiagramm charakterisiert hierbei die Obermenge von allen strukturell gültigen virtuellen

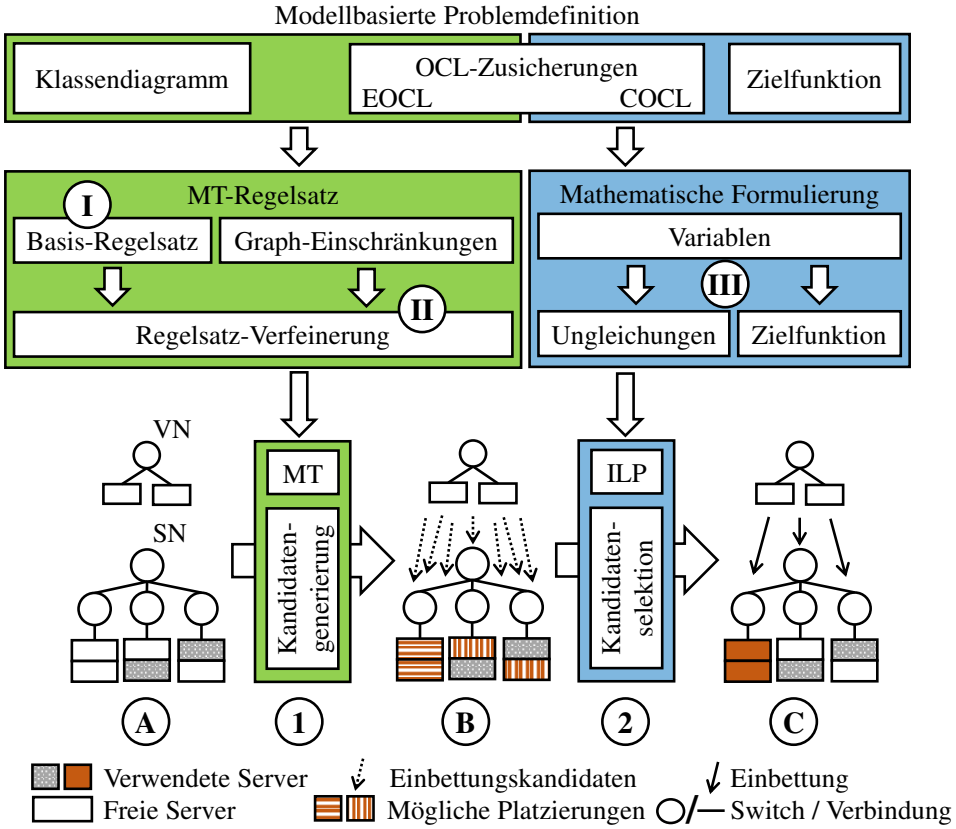


Abbildung 4.2: Schematische Darstellung der Konstruktionsmethodik

und Substratnetzwerken. Die OCL-Zusicherungen integrieren zusätzliche (strukturelle) Konsistenzeigenschaften in das Klassendiagramm, die eine gültige Einbettung erfüllen muss und die nicht durch das Klassendiagramm selbst ausgedrückt werden können (z.B. Anforderung V3: die aggregierten Ressourcen aller virtuellen Server auf einem Substratserver dürfen die Ressourcen des Substratservers nicht überbuchen). Die Zielfunktion, welche ebenfalls in OCL ausgedrückt wird, legt das Optimierungsziel für das VNE-Problem fest (z.B. Minimierung der Kommunikationskosten oder des Energieverbrauchs).

Die vorliegende Methodik transformiert nun die deklarative, modellbasierte Problemdefinition in eine korrekte und optimalitätserhaltende MdVNE-Konfiguration bestehend aus einem MT- und einem ILP-Anteil. Die MdVNE-Konfiguration ist hierbei korrekt in Bezug auf die modellbasierte Problemdefinition, wenn nach Schritt ② der Kandidatenselektion die Menge an Platzierungskandidaten gemäß dem Metamodell (Klassendiagramm und OCL-Zusicherungen) zulässig ist. Die MdVNE-Konfiguration erhält die Optimalität in Bezug auf die modellbasierte Problemdefinition, wenn diese korrekt ist und in der Menge der erzeugten Platzierungskandidaten mindestens ein Kandidat existiert, sodass alle anderen möglichen Platzierungen bezüglich des vorgegebenen Optimierungsziels nicht als besser bewertet werden können (Kandidatenselektion ②). Dazu unterteilt sich die hier vorgestellte per-Konstruktion-korrekte Methodik in drei Hauptschritte, welche auch in Abb. 4.2 als ①, Ⅱ und Ⅲ dargestellt sind. Diese Schritte transformieren das Metamodell (mit dem Klassendiagramm und den OCL-Zusicherungen)

und die Zielfunktion in eine MT- und ILP-Konfiguration für die Kandidatengenerierung (grüner Bereich) und die Kandidatenselektion (blauer Bereich).

Zunächst leiten wir in Schritt Ⓘ einen grundlegenden TGG-Regelsatz ab, um alle möglichen validen Modellinstanzen für das Klassendiagramm abzudecken. Hierbei nutzen wir den Ansatz von Kehrer et al. [53], um konstruktiv sicherzustellen, dass alle möglichen virtuellen Netzwerke, Substratnetzwerke und deren Kombinationen, die konform zum Klassendiagramm sind, durch diese grundlegenden TGG-Regeln konstruiert werden können. Danach, in Schritt Ⓘ, wird dieser grundlegende TGG-Regelsatz verfeinert, um möglichst viele OCL-Zusicherungen zu integrieren. Dazu unterteilen wir zuerst die OCL-Zusicherungen in EOCL [80], eine Untermenge von OCL für die Prädikatenlogik erster Stufe, sowie arithmetischen und Mengen-Operatoren, und in COCL (komplementäres OCL), eine Untermenge von OCL für die Prädikatenlogik zweiter Stufe. Die EOCL-Zusicherungen können dann in zusätzliche Vorbedingungen bzw. Graph-Einschränkungen für die grundlegenden TGG-Regeln transformiert werden, welche sicherstellen, dass bei jeder Anwendung dieser modifizierten TGG-Regeln die EOCL-Zusicherungen nicht verletzt werden. Somit wird garantiert, dass beim Schritt der Kandidatengenerierung nur Platzierungskandidaten verworfen werden, die mindestens eine EOCL-Zusicherung verletzen. Dazu nutzen wir die Konstruktionsmethodik von Radke et al. [80], welche eine Erweiterung zur Methodik von Heckel et al. [47] darstellt. Bei der Konstruktionsmethodik von Radke et al. wird jede EOCL-Zusicherung zunächst in eine (verschachtelte) Graph-Einschränkung transformiert [43], welche dann in eine schwächste Vorbedingung zur Ausführung für eine betrachtete TGG-Regel transformiert wird [21]. Diese Vorbedingung ist notwendig und hinreichend, um sicherzustellen, dass die nun mit zusätzlichen Vorbedingungen angereicherten TGG-Regeln, welche die MT-Konfiguration bilden, die Graph-Einschränkungen bewahren. Der letzte Schritt Ⓘ transformiert die Zielfunktion und die COCL-Zusicherungen in die ILP-Konfiguration. Dabei wird jede COCL-Zusicherung in eine Nebenbedingung in der ILP-Formulierung übersetzt. Die Transformation der Zielfunktion ist meist sehr direkt möglich, da die Optimierungsfunktion normalerweise mit ILP-Variablen formuliert wird. Nachdem alle drei Konstruktionsschritte durchlaufen sind, kann die erstellte MdVNE-Konfiguration als Eingabe für den MdVNE-Ansatz verwendet werden (siehe Abschnitt 4.1).

4.3.1 *Erstellung der MT-Konfiguration*

Für die Erstellung der MT-Konfiguration sind zwei Hauptschritte notwendig:

1. Erstellung von grundlegenden TGG-Regeln (Ⓘ) und
2. die Verfeinerung bzw. Anreicherung dieser TGG-Regeln, um möglichst viele OCL-Zusicherungen in diesen Regelsatz zu integrieren (Ⓘ).

Wir beginnen daher mit der Konstruktion der grundlegenden TGG-Regeln, gehen danach auf die Generierung der Graph-Einschränkungen ein und präsentieren zum Schluss die Verfeinerung der TGG-Regeln, wodurch eine MT-Konfiguration für MdVNE entsteht.

4.3.1.1 *Konstruktion grundlegender TGG-Regeln*

Zur Erstellung der grundlegenden TGG-Regeln R_{MM} nutzen wir eine Konstruktionsmethodik von Kehrer et al. [53], um einen TGG-Regelsatz für die Menge von Operationen abzuleiten mit deren Hilfe alle gültigen Modellinstanzen $\mathfrak{L}(MM)$ eines gegebenen Metamodells MM erzeugt werden können. Da beim statischen VNE-Problem nur neue Elemente hinzugefügt werden, reicht es aus nur die objekterzeugenden Regeln zu berücksichtigen. Die objektlöschenden, -verschiebenden oder -verändernden Transformationsregeln werden daher vernachlässigt. Da das Substratnetzwerk unabhängig von den virtuellen Netzwerken und deren Einbettungen erstellt wird, konstruieren wir zunächst einen Regelsatz zur Erstellung eines vollständigen Substratnetzwerkes. Danach werden die Regeln zur Erstellung der virtuellen Netzwerke und deren Einbettungen bzw. Platzierungskandidaten konstruiert, wobei das Substratnetzwerk als notwendiger Kontext in den TGG-Regeln dient.

Im Folgenden stellen wir die aus dem Klassendiagramm (siehe Abb. 3.2) abgeleiteten grundlegenden TGG-Regeln vor. In Abb. 4.3 sind die sechs grundlegenden Regeln zur Erzeugung des Substratnetzwerkes abgebildet. In diesen Regeln werden weder die virtuellen Netzwerke noch das Korrespondenzmodell erstellt. In Abb. 4.3a, der Axiomregel, wird ein Substratnetzwerk erstellt, welches als Kontextelement für alle anderen Regeln dient. In Abb. 4.3b und 4.3c wird ein Substratswitch bzw. ein Substratserver erstellt und mit dem Substratnetzwerk verbunden. Abbildung 4.3d erzeugt einen Pfad der Länge Null mit identischen Start- und Zielknoten im Substratnetzwerk. In den beiden Regeln in Abb. 4.3e und 4.3f werden weitere Pfade der Länge eins bzw. zwei erzeugt, wodurch Pfade über maximal drei Substratknoten erstellt werden können. Abhängig von der Topologie des Substratnetzwerkes (z.B. switch- oder serverzentriert) können weitere Regeln analog zu Abb. 4.3f benötigt werden, um Pfade der Länge drei, vier oder mehr zu realisieren. Da in dieser Arbeit jede virtuelle Verbindung als Start- bzw. Zielelement einen Server und Switch besitzt und gebräuchliche Rechenzentren in einer 3-Tier-Topologie (z.B. Fat-Tree) [7] aufgebaut sind, genügt die Betrachtung der Pfade bis zu einer Länge von drei, da der längste Pfad zwischen einem Server und einem Switch in dieser Topologie drei Hops beträgt.

Abbildung 4.4 zeigt die grundlegenden Regeln zur Generierung der virtuellen Elemente zusammen mit den entsprechenden Einbettungen bzw. Platzierungskandidaten in ein bestehendes Substratnetzwerk. Die gemeinsame Erzeugung der virtuellen Elemente und deren Korrespondenzelemente ist notwendig, um sicherzustellen, dass Multiplizitäten der `virtuell`$*$ bzw. `substrat`$*$ Assoziationen von einem virtuellen zu einem Substratelement auch eingehalten werden, da jedes virtuelle Element nur auf ein Substratelement platziert werden darf (siehe Abb. 3.2). Dabei entspricht der Erzeugung einer Korrespondenzelementinstanz der Erzeugung einer `virtuell`$*$- bzw. `substrat`$*$-Assoziation zwischen einem virtuellen und einem Substratelement. Die Regel in Abb. 4.4a (kurz R_{NzN}) erstellt hierzu ein virtuelles Netzwerk und bettet dieses in ein bestehendes Substratnetzwerk ein. In den Regeln Abb. 4.4b (kurz R_{SzK}) und Abb. 4.4c (kurz R_{SzS}) werden die virtuellen Switche bzw. Server erzeugt und auf einen (abstrakten) Substratknoten bzw. einen Substratserver platziert. Die Regel Abb. 4.4e (kurz R_{VzP}) erzeugt

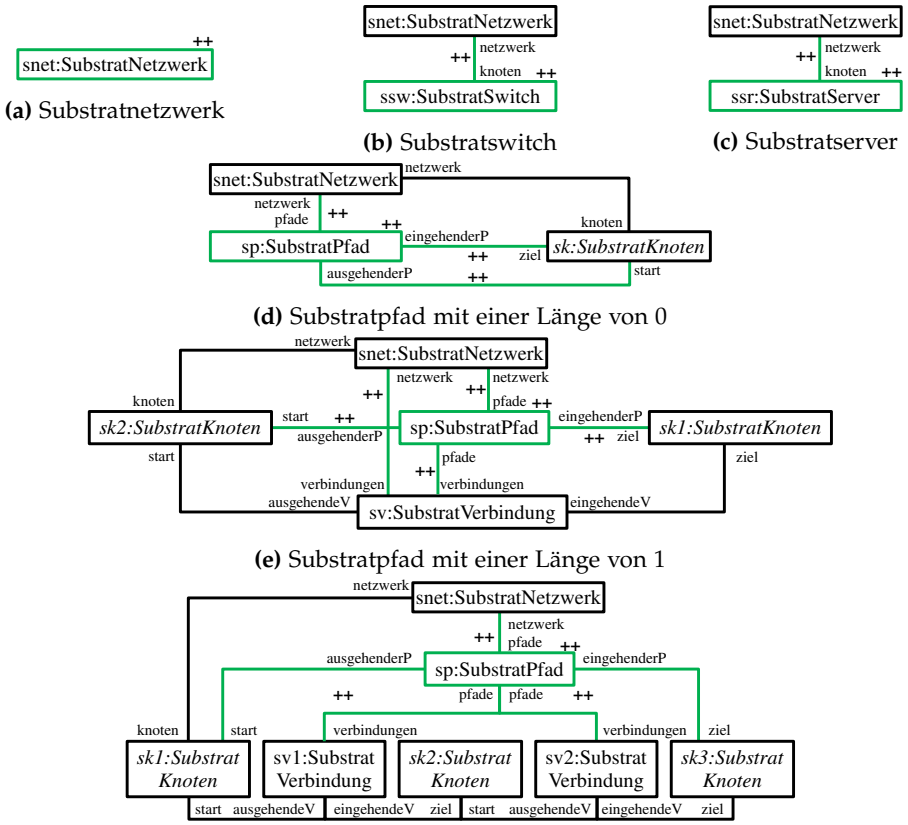
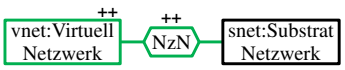
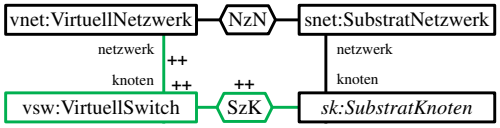


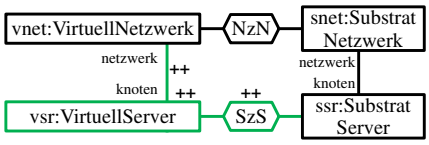
Abbildung 4.3: Basis-Regelsatz zur Erstellung des Substratnetzwerks und den enthaltenen Substratelementen



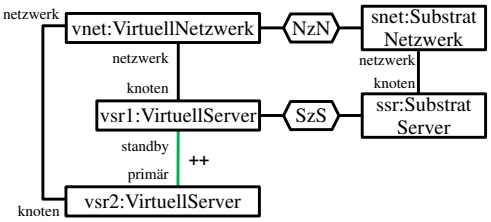
(a) Virtuelles Netzwerk und `NzN` (R_{NzN})



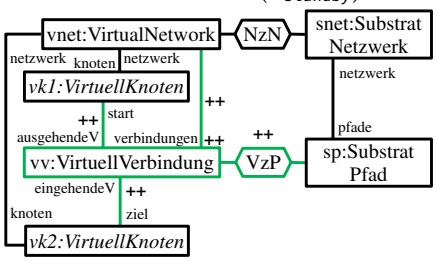
(b) Virtueller Switch und `SzK` (R_{SzK})



(c) Virtueller Server und `SzS` (R_{SzS})



(d) Virtueller primärer und Standby-Server ($R_{standby}$)



(e) Virtuelle Verbindung und `VzP` (R_{VzP})

Abbildung 4.4: Basis-Regelsatz zur Erstellung der virtuellen Netzwerke mit den enthaltenen virtuellen Elementen und den Platzierungen

eine virtuelle Verbindung und bettet diese in einem Substratpfad ein. Die Regel Abb. 4.4d (kurz $R_{standby}$) stellt in dieser Reihe eine Ausnahme dar, da kein neues virtuelles Element angelegt, sondern nur eine Assoziation zur Realisierung der Standby-Funktionalität zwischen dem virtuellen Server `vsr1` und dem Standby-Server `vsr2` erzeugt wird.

> **Beispiel 4.3** (Grundlegende TGG-Regeln)**.** Nach der Ausführung der grundlegenden TGG-Regeln kann das virtuelle und das Substratnetzwerk einschließlich der Einbettungen aus Abb. 3.3a erzeugt werden.

4.3.1.2 *Verfeinerung des Regelsatzes*

Durch die im vorhergehenden Abschnitt konstruierten grundlegenden TGG-Regeln (Konstruktionsschritt Ⓘ), welche in Abb. 4.3 und 4.4 abgebildet sind, können alle virtuellen und Substratnetzwerke und deren Einbettungen bzw. Platzierungskandidaten gemäß dem Klassendiagramm aus Abb. 3.2 erstellt werden. Dennoch können die daraus resultierenden Modelle die zusätzlich spezifizierten OCL-Zusicherungen (siehe Abschnitt 3.3) verletzen, was zu ungültigen Netzwerken und Einbettungen führen kann. So kann beispielsweise durch das Anwenden der TGG-Regel R_{SzS} (Abb. 4.4c) ein virtueller Server auf einem Substratserver platziert werden, obwohl die Rechenkapazität (*cpu*) des Substratservers nicht ausreicht, um den virtuellen Server darauf zu betreiben (d.h. `VirtuellServer`.*cpu* > `SubstratServer`.*cpu*).

Zur Reduzierung von (möglichen) Platzierungskandidaten und ungültigen Netzwerkinstanzen werden die grundlegenden TGG-Regeln weiter verfeinert, indem so viele OCL-Zusicherungen wie möglich aus dem Satz aller OCL-Zusicherungen aus Abschnitt 3.3 durch den verfeinerten Regelsatz sichergestellt werden (Konstruktionsschritt (II)). Hierfür verwenden wir eine bestehende statische Konstruktionsmethodik, die OCL-Zusicherungen in Anwendungsbedingungen von Modelltransformationsregeln überführt [19, 47, 80]. Die Integration dieser Anwendungsbedingungen in den TGG-Regelsatz verhindert, dass TGG-Regeln in Kontexten angewendet werden, in denen die Regelanwendung die jeweilige OCL-Zusicherung verletzen würde. Die Methodik in [80] unterstützt hierbei nur die Transformation von EOCL, einer Teilmenge der vollständigen OCL-Spezifikation [38]. Im Allgemeinen umfasst hierbei EOCL die Elemente von OCL, die auf der Prädiktenlogik erster Stufe aufbauen (z.B. Aussagenlogik, Prädikate und Quantoren über elementwertigen Variablen), sowie arithmetischen und Mengen-Operatoren. Einen umfassenden Überblick über EOCL und deren Elemente ist in [80] zu finden. Zur Beschreibung des VNE-Problems werden in der vorliegenden Arbeit OCL-Zusicherungen aus EOCL und der `iterate()` Operation (analog zur `sum()` Operation) in OCL verwendet, wodurch wir bei der Verfeinerung der grundlegenden TGG-Regeln nur die EOCL-Zusicherungen berücksichtigen und die weiteren notwendigen COCL-Zusicherungen, welche auch `iterate()` beinhalten, auf den Transformationsschritt (III) aus Abb. 4.2 verschieben.

Relaxation der OCL-Zusicherungen Im Folgenden werden die OCL-Zusicherungen relaxiert, da sich nur EOCL-Zusicherungen in Regelvorbedingungen übersetzen lassen und die Menge an (potentiell) ungültigen Platzierungskandidaten durch möglichst strenge EOCL-Zusicherungen verringert werden kann. Die Menge von OCL-Zusicherungen aus Abschnitt 3.3 besteht aus fünf EOCL-Zusicherungen C_{start}, C_{ziel}, C_{a1}, C_{a2}, C_{a3} und vier COCL-Zusicherungen C_{cpu}, C_{ram}, C_{hdd}, C_{bb}. Für die COCL-Zusicherungen, welche das Überbuchen von Ressourcen für Server und Verbindungen verbieten, werden relaxierte EOCL-Zusicherungen (ROCL-Zusicherungen) abgeleitet, da für jede einzelne Einbettung bzw. für jeden Platzierungskandidaten mindestens gelten muss, dass die Ressourcen des virtuellen Elements kleiner oder gleich der Ressourcen des Substratelements sind. Diese Menge dieser konkreten ROCL-Zusicherungen wird mit C_{ROCL} bezeichnet.

Beispiel 4.4 (Relaxation der OCL-Zusicherungen)**.** Anhand von C_{cpu} erläutern wir die Grundidee dieser Relaxation. Für einen gegebenen Substratserver und eine Menge von virtuellen Servern, die auf diesen Substratserver platziert werden, muss gelten, dass die Summe der geforderten Rechenkapazitäten (*cpu*) aller virtuellen Server kleiner oder gleich der durch den Substratserver zur Verfügung gestellten Rechenkapazitäten sein muss. Daher muss mindestens gelten, dass die Rechenkapazität jedes einzelnen virtuellen Servers kleiner oder gleich als die verfügbare Rechenkapazität des Substratservers ist.

Eine *OCL-Zusicherung C_B relaxiert dabei eine OCL-Zusicherung C_A*, wenn jedes Modell, das C_{A} erfüllt, auch C_{B} erfüllt. Umgekehrt verletzt jedes Modell, wel-

Tabelle 4.1: Transformation der OCL-Zusicherungen in ROCL-Zusicherungen

Original-Zusicherung	**Typ**	**ROCL-Zusicherung für ⓘⓘ**
C_{cpu}	COCL	C_{cpuR}
C_{ram}	COCL	C_{ramR}
C_{hdd}	COCL	C_{hddR}
C_{bb}	COCL	C_{bbR}
C_{start}	EOCL	–
C_{ziel}	EOCL	–
C_{a1}	EOCL	–
C_{a2}	EOCL	–
C_{a3}	EOCL	–

ches C_A nicht erfüllt, auch C_B. Somit wird kein möglicher Platzierungskandidat durch die relaxierten Varianten der COCL-Zusicherungen ausgeschlossen. Aufgrund dieser Definition unterscheiden wir drei Fälle zur weiteren Behandlung einer OCL-Zusicherung C_A aus der modellbasierten Problemdefinition:

1. Wenn C_A eine EOCL-Zusicherung darstellt, verfeinern wir die TGG-Regeln für die Kandidatengenerierung ① auf der Basis von C_A und können danach C_A bei der Kandidatenselektion ② vernachlässigen.

2. Wenn C_A eine COCL-Zusicherung darstellt und wir eine (triviale) Relaxation C_B für C_A finden, welche als EOCL-Zusicherung ausgedrückt werden kann, dann wird die relaxierte Zusicherung C_B in die Kandidatengenerierung ① integriert und die Originalzusicherung C_A durch die Kandidatenselektion ② sichergestellt.

3. Wenn C_A eine COCL-Zusicherung darstellt, für welche keine EOCL-Relaxation abgeleitet werden kann, wird diese Zusicherung (C_A) nur durch die Kandidatenselektion ② gewährleistet.

Im vorliegenden Szenario können wir die vier COCL-Zusicherungen C_{cpu}, C_{ram}, C_{hdd}, C_{bb} durch die folgenden ROCL-Zusicherungen C_{cpuR}, C_{ramR}, C_{hddR}, C_{bbR} ausdrücken.

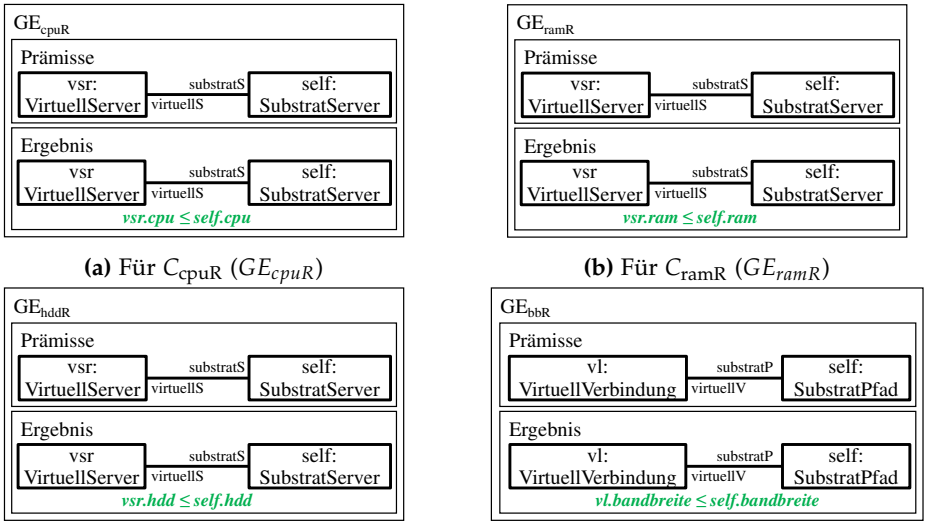
context SubstratServer **inv** self.virtuellS → forAll(vs|vs.cpu ≤ self.cpu) (C_{cpuR})
context SubstratServer **inv** self.virtuellS → forAll(vs|vs.ram ≤ self.ram) (C_{ramR})
context SubstratServer **inv** self.virtuellS → forAll(vs|vs.hdd ≤ self.hdd) (C_{hddR})
context SubstratPfad **inv** self.virtuellV →
forAll(vl|vl.bandbreite ≤ self.bandbreite) (C_{bbR})

Tabelle 4.1 fasst die ursprünglichen OCL-Zusicherungen aus Abschnitt 3.3 und deren Relaxationen (falls erforderlich) zusammen.

Ableitung der Graph-Einschränkungen Im Folgenden beschreiben wir, wie die Menge von EOCL-Zusicherungen, welche die im vorherigen Abschnitt erstellten ROCL-Zusicherungen beinhaltet, in eine Menge von Anwendungsbedingungen für die grundlegenden TGG-Regeln transformiert wird. Hierzu verwenden wir die Konstruktionsmethodik von Radke et al. [80] und Heckel et al. [47]. Das Ziel in diesem Schritt besteht hierbei, einen TGG-Regelsatz abzuleiten, der die Konsistenz zu allen neun EOCL-Zusicherungen und ROCL-Zusicherungen (siehe Tabelle 4.1) gewährleistet. Das bedeutet, dass wir in diesem Konstruktionsschritt alle weiteren COCL-Zusicherungen ausschließen. Eine TGG-Regel ist dabei *konsistenzerhaltend*, wenn ein Modell, dass die angegebenen Einschränkungen erfüllt, in ein Modell transformiert wird, das diese Einschränkungen ebenfalls erfüllt. Durch den Fokus auf die Konsistenzerhaltung können wir zusätzlich analysieren, inwieweit eine einzelne Anwendung einer TGG-Regel hierbei zu Problemen führen kann. Dabei wird die Konsistenz nur erhalten, wenn das ursprüngliche Modell der Netzwerke bereits konsistent ist. Dies ist im vorliegenden Szenario der Fall, da das ursprüngliche Modell der Netzwerke keine Einbettungen enthält. Zusätzlich gehen wir davon aus, dass jedes virtuelle Netzwerk und das Substratnetzwerk für sich genommen konform zum Metamodell sind (kann gegebenenfalls vorab zusätzlich geprüft werden).

Zunächst übersetzen wir die EOCL-Zusicherungen mithilfe der Methodik von Radke et al. [80] in verschachtelte Graph-Einschränkungen. Für das in dieser Arbeit verwendete Beispiel ist eine Verschachtelungstiefe von eins für die Graph-Einschränkungen ausreichend. Zur Darstellung dieses Untertyps von verschachtelten Graph-Einschränkungen nutzen wir *Prämisse-Ergebnis-Zusicherungen* [47], welche wie folgt, definiert sind: Eine *Graph-Einschränkung* besteht hierbei aus einem Muster, welches die *Prämisse* darstellt (*Prämissen-Muster*), und einer (möglicherweise leeren) Menge von Mustern, die die *Ergebnisse* dieser Graph-Einschränkung beschreiben (*Ergebnis-Muster*). Ein *Muster* besteht hierbei aus einem Mustergraphen mit Objekt- und Verbindungsvariablen und relationalen Attributeinschränkungen. Eine *Übereinstimmung eines Musters in einem Modell* bildet hierbei die durch den Mustergraphen repräsentierten Variablen so im Modell ab, dass alle Attributeinschränkungen erfüllt sind. Dabei erweitert jedes Ergebnis-Muster einer Graph-Einschränkung das Prämissen-Muster. Ein *Muster m_A erweitert dabei ein Muster m_B*, wenn der Mustergraph von m_A ein Untergraph des Mustergraphen von m_B darstellt und alle Attributeinschränkungen von m_B die Attributeinschränkungen von m_A implizieren. Eine *positive Graph-Einschränkung* hat mindestens ein Ergebnis-Muster, wobei eine *negative Graph-Einschränkung* kein Ergebnis-Muster besitzt.

Beispiel 4.5 (Positive Graph-Einschränkung)**.** Die positive Graph-Einschränkung GE_{cpuR} abgleitet aus der ROCL-Zusicherung C_{cpuR} in Abb. 4.5a stellt eine positive Graph-Einschränkung dar, da diese neben dem Prämissen-Muster auch ein Ergebnis-Muster besitzt. Das Prämissen-Muster beschreibt hierbei einen virtuellen Server (`vsr`) der auf einem Substratserver (`self`) platziert ist. Das Ergebnis-Muster erweitert das Prämissen-Muster um eine zusätzliche Attributeinschränkung (`vsr.cpu` $\leq$ `self.cpu`), welche sicherstellt, dass die ge-



(a) Für C_{cpuR} (GE_{cpuR}) **(b)** Für C_{ramR} (GE_{ramR})

(c) Für C_{hddR} (GE_{hddR}) **(d)** Für C_{bbR} (GE_{bbR})

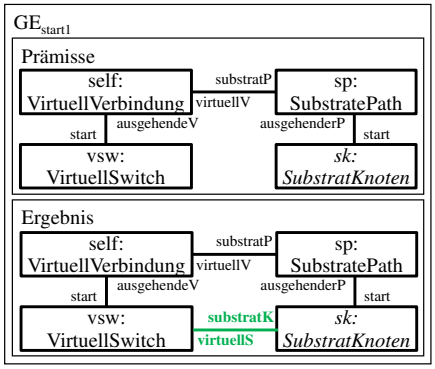
Abbildung 4.5: Graph-Einschränkung für die ROCL-Zusicherungen C_{cpuR}, C_{ramR}, C_{hddR} und C_{bbR}

> forderte Rechenkapazität des virtuellen Servers (`vsr.cpu`) nicht größer als die durch den Substratserver verfügbare Rechenkapazität (`self.cpu`) ist.

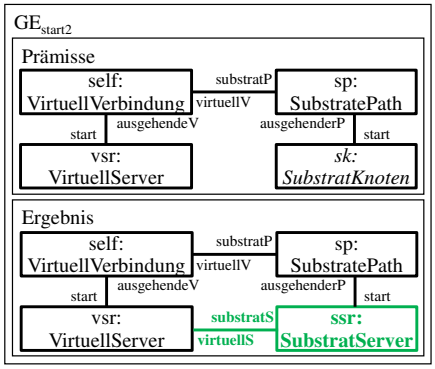
Ein *Modell erfüllt solche Graph-Einschränkungen*, wenn jede Übereinstimmung eines Prämissen-Musters dieser Graph-Einschränkung in eine Übereinstimmung von mindestens einem Ergebnis-Muster erweitert werden kann (positive Graph-Einschränkung). Falls die Menge an Ergebnis-Mustern eine leere Menge darstellt, dann darf somit das Prämissen-Muster nicht im Modell vorhanden sein, wodurch diese Graph-Einschränkung auch negative Graph-Einschränkung genannt wird.

> **Beispiel 4.6** (Erfüllung einer Graph-Einschränkung)**.** Ein Modell erfüllt GE_{cpuR} dann, und nur dann, wenn jeder virtuelle Server, der auf einem Substratserver platziert ist, die Rechenkapazität des jeweiligen Substratservers nicht überschreitet.

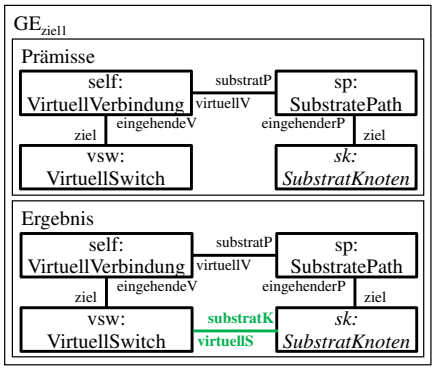
Im Folgenden transformieren wir mithilfe der bei Radke et al. [80] präsentierten Methodik nun aus den neun EOCL-Zusicherungen aus Tabelle 4.1 Graph-Einschränkungen, welche in den Abb. 4.5 bis 4.7 dargestellt sind. Wenn möglich verwenden wir in den Graph-Einschränkungen die Variablennamen aus den OCL-Zusicherungen, um die Entsprechungen zwischen den Graph-Einschränkungen und den OCL-Zusicherungen zu verdeutlichen. So wird beispielsweise die Objektvariable, die dem Kontext der OCL-Zusicherung entspricht, immer mit `self` benannt. In Abb. 4.5 sind die vier abgeleiteten Graph-Einschränkungen GE_{cpuR}, GE_{ramR}, GE_{hddR} und GE_{bbR} abgebildet, welche den ROCL-Zusicherungen C_{cpuR}, C_{ramR}, C_{hddR} und C_{bbR} entsprechen. Jede dieser Graph-Einschränkungen stellt dabei eine positive Graph-Einschränkung mit einem Prämissen-Muster und einem Ergebnis-Muster dar, welche sich lediglich in den zusätzlichen Attributeinschränkungen im Ergebnis-Muster unterscheiden.



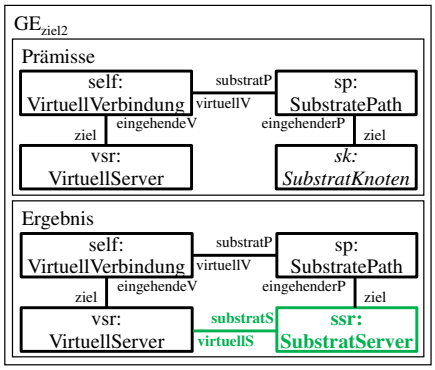
(a) Für den *dann-Zweig* von C_{start} (GE_{start1})



(b) Für den *else-Zweig* von C_{start} (GE_{start2})



(c) Für den *dann-Zweig* von C_{ziel} (GE_{ziel1})

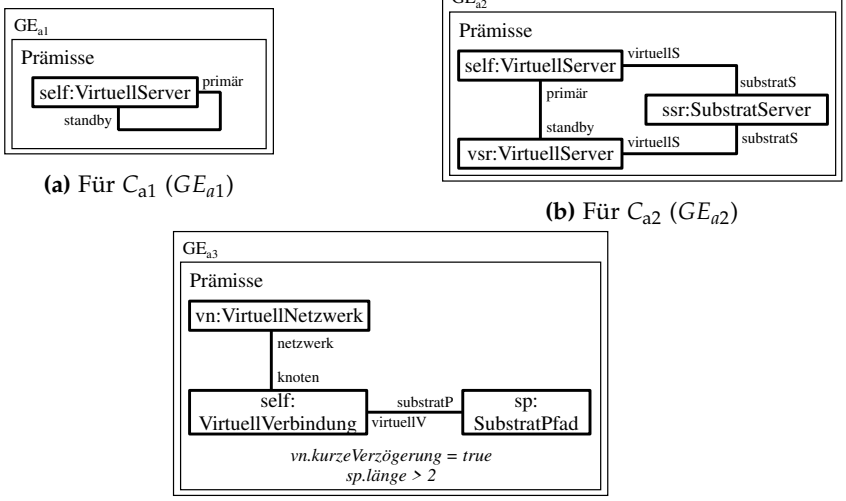


(d) Für den *else-Zweig* von C_{ziel} (GE_{ziel2})

Abbildung 4.6: Graph-Einschränkungen für die EOCL-Zusicherungen C_{start} und C_{ziel}

Die vier Graph-Einschränkungen in Abb. 4.6 entsprechen den beiden Zusicherungen C_{start} und C_{ziel}, die fordern, dass ein virtueller Start- bzw. Zielknoten einer virtuellen Verbindung auf den Start- bzw. Zielknoten des Substratpfades platziert wird, auf den die virtuelle Verbindung eingebettet ist. Die erhöhte Anzahl an Graph-Einschränkungen kommt durch die (`if-then-else`-)Bedingung zustande, die abhängig von der konkreten Klasse des Start- bzw. Zielknotens der virtuellen Verbindung (`self`) ist. Die Graph-Einschränkungen GE_{start1} und GE_{ziel1} repräsentieren hierbei die Situation, in welcher der virtuelle Start- bzw. Zieknoten einen virtuellen Switch (`vsw`) darstellt (`self.start.oclAsType(VirtuellSwitch)`). Die beiden anderen Graph-Einschränkungen GE_{start2} und GE_{ziel2} decken den Fall ab, dass die Start- bzw. Zielknoten einen virtuellen Server (`vsr`) darstellen (`self.start.oclAsType(VirtuellServer)`).

Zum Schluss entsprechen die drei Graph-Einschränkungen GE_{a1}, GE_{a2} und GE_{a3} in Abb. 4.7 den EOCL-Zusicherungen C_{a1}, C_{a2} und C_{a3}. Diese Graph-Einschränkungen sind negative Graph-Einschränkungen, da sie kein Ergebnis-Muster besitzen. Daher dürfen die Prämissen-Muster zu keiner Zeit im Modell vorhanden sein. In C_{a1} (GE_{a1}) wird gefordert, dass es keinen virtuellen Server (`self`) gibt, der sich selbst als Standby-Server besitzt. C_{a2} (GE_{a2}) repräsentiert die Zusicherung, dass kein virtueller Server (`self`) und dessen Standby-Server (`vsr`) auf demselben Substratserver (`ssr`) platziert sind. In C_{a3} (GE_{a3}) wird sichergestellt, dass keine virtuelle Verbindung (`self`) in einem virtuellen Netzwerk, welches ei-



(a) Für C_{a1} (GE_{a1})

(b) Für C_{a2} (GE_{a2})

(c) Für C_{a3} (GE_{a3})

Abbildung 4.7: Graph-Einschränkungen für die EOCL-Zusicherungen C_{a1}, C_{a2} und C_{a3}

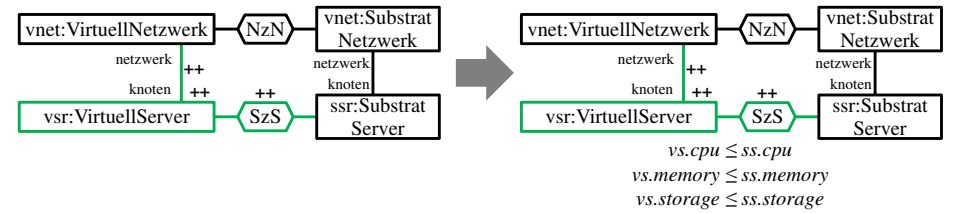
Tabelle 4.2: Verfeinerung des TGG-Basisregelsatzes (Abb. 4.4) basierend auf den EOCL-Zusicherungen

	C_{cpuR}	C_{ramR}	C_{hddR}	C_{start}	C_{ziel}	C_{bbR}	C_{a1}	C_{a2}	C_{a3}
R_{NzN}	–	–	–	–	–	–	–	–	–
R_{SzS}	+	+	+	–	–	–	–	–	–
R_{SzK}	–	–	–	–	–	–	–	–	–
$R_{standby}$	+	+	+	–	–	–	–	+	–
R_{VzP}	–	–	–	+	+	+	–	–	+

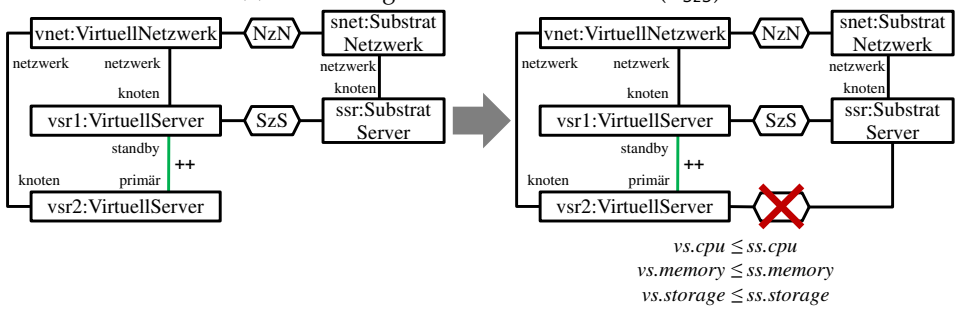
ne kurze Verzögerung fordert (`vn.kurzeVerzögerung = true`), auf einen Substratpfad (`sp`) mit einer Länge von mehr als 2 Hops (`sp.länge > 2`) platziert wird.

Integration der Graph-Einschränkungen in die TGG-Regeln Der nächste Schritt besteht nun darin, jede einzelne TGG-Regel mit den Graph-Einschränkungen zu verfeinern. Dabei ist jede Iteration zur Verfeinerung der Regeln unabhängig von anderen Verfeinerungsiterationen. Da es fünf grundlegende TGG-Regeln, die Einbettungen bzw. Platzierungskandidaten erzeugen, (siehe Abb. 4.4) und elf Graph-Einschränkungen (siehe Abb. 4.5 bis 4.7) gibt, müssen nun 55 Verfeinerungsiterationen vorgenommen werden. Tabelle 4.2 gibt einen Überblick über die Ergebnisse dieses konstruktiven Schrittes, welche im Folgenden näher diskutiert werden. Dabei sind die Iterationen für die Graph-Einschränkungen von GE_{start1} und GE_{start2} sowie für GE_{ziel1} und GE_{ziel2} in den Spalten für C_{start} bzw. C_{ziel} zusammengefasst. Dadurch verringert sich die Anzahl der Einträge auf 45. Falls durch diese Verfeinerung Änderungen an den TGG-Regeln durchgeführt wurden, ist dies durch eine +-Markierung gekennzeichnet.

Die Grundidee bei einer Verfeinerungsiteration besteht darin, für ein gegebenes Regel-Einschränkungspaar alle Situationen zu identifizieren, in denen die unmo-



(a) Platzierung eines virtuellen Servers (R_{SzS})



(b) Hinzufügen eines Standby-Servers ($R_{standby}$)

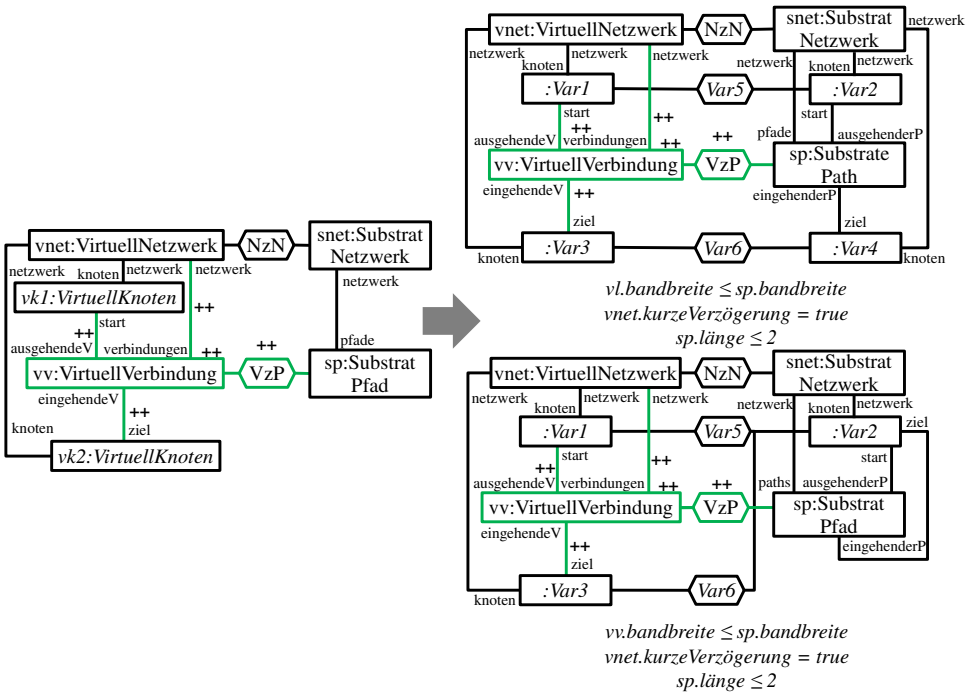
Abbildung 4.8: Verfeinerungen des TGG-Regelsatzes für Platzierungen von Servern

Tabelle 4.3: Variationen der MT Regel zur Platzierung eines virtuellen Links auf einem Substratpfad (Abb. 4.9)

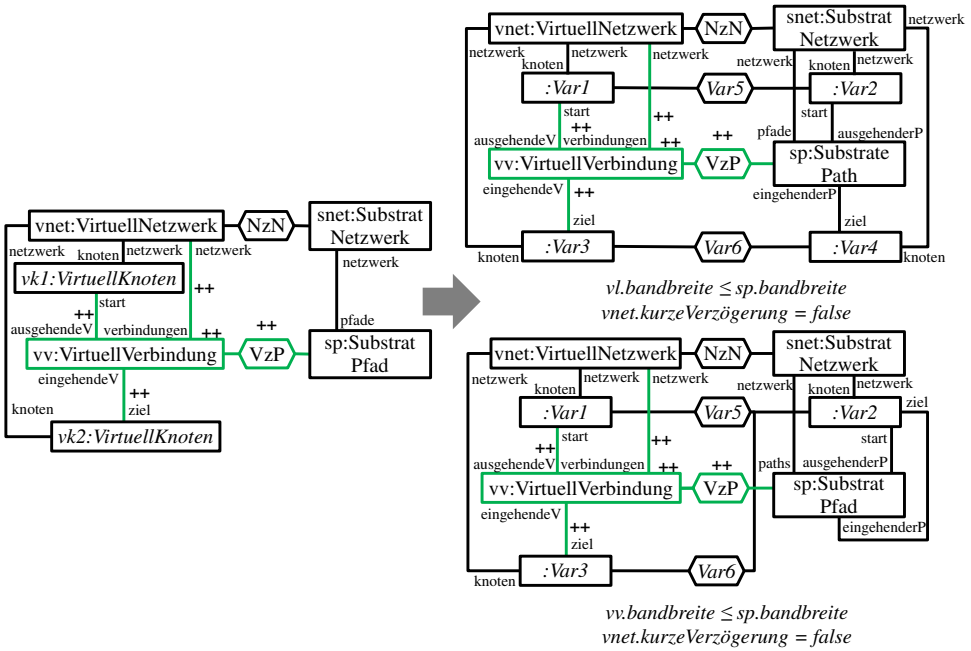
		Objektvariablen	
Var1	***Var2***	***Var3***	***Var4***
`VirtuellSwitch`	`VirtuellSwitch`	`SubstratKnoten`	`SubstratKnoten`
`VirtuellSwitch`	`VirtuellSwitch`	`SubstratKnoten`	–
`VirtuellSwitch`	`VirtuellServer`	`SubstratKnoten`	`SubstratServer`
`VirtuellSwitch`	`VirtuellServer`	`SubstratKnoten`	–
`VirtuellServer`	`VirtuellSwitch`	`SubstratServer`	`SubstratKnoten`
`VirtuellServer`	`VirtuellSwitch`	`SubstratServer`	–
`VirtuellServer`	`VirtuellServer`	`SubstratServer`	`SubstratServer`
`VirtuellServer`	`VirtuellServer`	`SubstratServer`	–

difizierte Regel diese Einschränkung verletzt. Danach wird jede Situation in eine schwächste Vorbedingung für diese Regel transformiert [21], die sicherstellt, dass die Regel dann, und nur dann anwendbar ist, wenn die Anwendung dieser Regel die gerade betrachtete Einschränkung nicht verletzt. Die Menge von allen generierten Vorbedingungen ist notwendig und hinreichend, um sicherzustellen, dass die verfeinerten TGG-Regeln die EOCL-Zusicherungen einhalten. Wir verzichten hier auf die Darstellung von Details des konstruktiven Ansatzes und verweisen stattdessen auf [47, 80].

In Abb. 4.8 und 4.9 wird gezeigt, wie drei der fünf grundlegenden TGG-Regeln verfeinert werden. Dabei werden die grundlegenden TGG-Regeln links vom Blockpfeil und die daraus resultierenden verfeinerten TGG-Regeln rechts davon dargestellt. Die TGG-Regeln R_{NzN} und R_{SzK} fehlen in diesen Abbildungen, da sie durch die Verfeinerung nicht verändert werden.



(a) Platzierung einer virtuellen Verbindung mit kurzer Verzögerungszeit (R_{VzP})



(b) Platzierung einer virtuellen Verbindung ohne kurze Verzögerungszeit (R_{VzP})

Abbildung 4.9: Verfeinerungen der TGG-Regeln für Platzierungen von Verbindungen

Abbildung 4.8a zeigt auf der linken Seite die grundlegende und auf der rechten Seite die verfeinerte TGG-Regel für R_{SzS} zur Einbettung eines virtuellen Servers in einem Substratserver. Hierbei wurde die grundlegende Regel R_{SzS} dreimal durch zusätzliche Attributeinschränkungen verfeinert, um sicherzustellen das die Rechenkapazität (GE_{cpuR}), der Arbeitsspeicher (GE_{ramR}) oder der Festplattenspeicher (GE_{hddR},) des Substratservers (`ssr`) durch die Einbettung des virtuellen Servers (`vsr`) nicht überschritten werden.

Die Verfeinerung der grundlegenden TGG-Regel $R_{standby}$ wird in Abb. 4.8b dargestellt, um eine neue Standby-Beziehung zwischen einem virtuellen (Standby-)Server (`vsr2`) und einem (primären) Server (`vsr1`) hinzuzufügen. Der zukünftige Standby-Server ist hierbei bereits eingebettet. Die drei Ressourceneinschränkungen GE_{cpuR}, GE_{ramR} und GE_{hddR} werden analog zur vorherigen Verfeinerung aus Abb. 4.8a durch drei zusätzliche Attributeinschränkungen integriert. Weiter wird die Regel $R_{standby}$ für die Graph-Einschränkung GE_{a2} durch eine negative Anwendungsbedingung verfeinert. Dies wird durch die rot durchgestrichene Korrespondenzvariable verdeutlicht. Eine *negative Anwendungsbedingung* schränkt hierbei die Anwendbarkeit einer Regel ein, da gefordert wird, dass eine Variable nicht existieren darf. In diesem Fall darf der virtuelle primäre Server `vrs2` nicht auf dem Substratserver `ssr` platziert sein, wenn der Standby-Server `vsr1` auf denselben Substratserver `ssr` eingebettet ist. Bei der Graph-Einschränkung GE_{a1} ist eine weitere Verfeinerung nicht notwendig, da durch die injektive Mustererkennung sichergestellt wird, dass der primäre und der Standby-Server auf unterschiedliche Substratserver im Netzwerk abgebildet werden.

In Abb. 4.9 wird die Verfeinerung der grundlegenden TGG-Regel R_{VzP}, der Platzierung einer virtuellen Verbindung auf einen Substratpfad, abgebildet. Durch GE_{a3} werden zwei sehr ähnliche Varianten gebildet, wobei im ersten Fall das virtuelle Netzwerk eine kurze Verzögerung fordert (siehe Abb. 4.9a) und im zweiten Fall keine kurze Verzögerung (siehe Abb. 4.9b) garantiert werden muss. Dazu werden zusätzliche Attributeinschränkungen in die grundlegende TGG-Regel integriert, wodurch einmal sichergestellt wird, dass, wenn eine kurze Verzögerung von einem virtuellen Netzwerk `vnet` gefordert wird (`vnet.kurzeVerzögerung = true`), auch die Länge des Substratpfades `sp` maximal zwei beträgt (`sp.länge` $\leq$ `2`). Zusätzlich erhalten wir durch die Verfeinerung von GE_{start1}, GE_{start2}, GE_{ziel1} und GE_{ziel2} für jede zuvor vorgestellte Variante (Abb. 4.9a und Abb. 4.9b) weitere acht zusätzliche Anwendungsbedingungen und somit acht verfeinerte Varianten der TGG-Regel, insgesamt also 16 verfeinerte TGG-Regeln. Tabelle 4.3 gibt einen Überblick über diese acht Varianten, welche durch die unterschiedlichen Klassen (Server oder Switch) der beteiligten virtuellen Start- oder Zielknoten entstehen. Betrachtet man nur die verschiedenen Möglichkeiten der virtuellen Start- und Zielknoten (virtueller Server oder Switch) ergeben sich nur vier Varianten, welche in den Kombinationen von `Var1`, `Var2` und `Var3` in Tabelle 4.3 dargestellt sind. Zusätzlich muss noch die Unterscheidung zwischen Substratpfaden der Länge null, deren Start- und Zielknoten identisch sind, und Substratpfaden mit einer Länge von mindestens eins, deren Start- und Zielknoten unterschiedlich sind, berücksichtigt werden. Der erste Fall ist unten rechts in den Abb. 4.9a und 4.9b und der zweite Fall oben rechts abgebildet. Die Korrespondenzvariablen `Var5`

und `Var6` lassen sich aus den Klassen von `Var1` und `Var2` bzw. `Var3` und `Var4` ableiten. Wenn beispielsweise `Var1` einen virtuellen Switch repräsentiert, dann ist die Korrespondenzvariable `SzK`, ansonsten `SzS`. Zusätzlich wird die Graph-Einschränkung GE_{bbR} durch eine zusätzliche Attributeinschränkung, die sicherstellt, dass die vorhandene Bandbreite nicht durch die virtuelle Verbindung überbucht wird, verfeinert. Die TGG-Regeln für Substratpfade der Länge Null bilden einen Sonderfall, da angenommen wird, dass ein Substratserver einen sehr großen, aber endlichen Wert besitzt, da eine serverinterne Kommunikation als sehr schnell angenommen wird.

4.3.2 *Erstellung der ILP-Konfiguration*

Durch die im vorherigen Abschnitt systematisch erstellte MT-Konfiguration können im Schritt der Kandidatengenerierung bei MdVNE alle Platzierungskandidaten generiert werden, welche garantiert alle EOCL-Zusicherungen (inklusive der ROCL-Zusicherungen) erfüllen. Das bedeutet, dass die ILP-Konfiguration im Schritt zur Kandidatenselektion (nur) sicherstellen muss, dass die MT-Konfiguration und die zusätzlichen, nicht durch die MT-Konfiguration abgedeckten COCL-Zusicherungen, eingehalten werden und nur optimale Platzierungskandidaten in Bezug auf die Zielfunktion erhalten bleiben. Die Generierung der ILP-Formulierung ist dazu (zur Laufzeit) in drei Schritte unterteilt:

1. Generierung der ILP-Formulierung auf Grundlage der MT-Konfiguration,
2. Generierung der ILP-Formulierung auf Grundlage der COCL-Zusicherungen und
3. Generierung der ILP-Formulierung für die Zielfunktion.

4.3.2.1 *ILP-Formulierung für die MT-Konfiguration*

Basierend auf der MT-Konfiguration müssen ILP-Formulierungen abgeleitet werden, um die grammatikalischen Eigenschaften der TGG-Regeln zu gewährleisten. Dazu verwenden wir einen generischen Ansatz zur Konsistenzprüfung mithilfe von TGG-Regeln [63], der im Programm eMoflon [62] enthalten ist. Hierbei werden zusätzliche ILP-Formulierungen zur Laufzeit generiert, sodass die TGG-Spezifikation [89] eingehalten wird. Dazu werden für alle Platzierungskandidaten (`SzK`, `SzS` und `VzP`), die durch die Ausführung der TGG-Regeln erzeugt werden, eindeutige ILP-Variablen erstellt, die notwendigen ILP-Ungleichungen generiert und diese zur ILP-Formulierung hinzugefügt. Im Folgenden werden die unterschiedlichen Platzierungsmöglichkeiten (virtueller Switch, Server oder virtuelle Verbindung) näher erläutert. Die dabei erstellten ILP-Ungleichungen sind dabei in Tabelle 4.4 zusammengefasst.

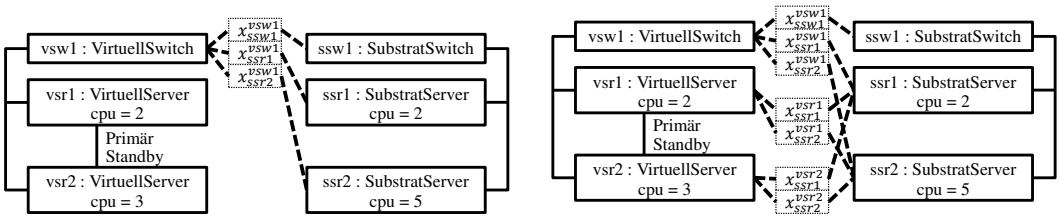
Platzierung eines virtuellen Switches Die TGG-Regel in Abb. 4.4b erzeugt für jede mögliche Platzierung eines virtuellen Switch $i \in N^S(Sw_i = 1)$ auf einen Substratknoten $u \in N^S$ einen Platzierungskandidaten szk_u^i. Für jeden dieser

Tabelle 4.4: Zusätzlich abgeleitete ILP-Formulierungen für die (verfeinerten) TGG-Regeln

TGG-Regel	ILP-Formulierungen abgeleitet von den TGG-Regeln
Abb. 4.4b	$\forall i \in N^V, Sw_i = 1 \mid \sum_{u=1}^{N^S} x_u^i \leq 1$
Abb. 4.8a	$\forall i \in N^V, Sr_i = 1 \mid \sum_{u=1}^{N^S, Sr_u=1} x_u^i \leq 1$
Abb. 4.9a und 4.9b	$\forall i,j \in N^V \mid \sum_{u=1}^{N^S} \sum_{v=1}^{N^S} y_{p_{u,v}}^{l_{i,j}} \leq 1$

Platzierungskandidaten erstellen wir eine ILP-Platzierungsvariable für eine Knotenplatzierung x_u^i, sodass genau eine eindeutige ILP-Variable x_u^i für jeden Platzierungskandidaten szk_u^i existiert (SzK→X). Um nun sicherzustellen, dass für jeden virtuellen Switch i genau ein Platzierungskandidat aus der Menge von möglichen Kandidaten szk_u^i ausgewählt wird, ist die folgende ILP-Ungleichung erforderlich.

$$\forall i \in N^V, Sw_i = 1 \mid \sum_{u=1}^{N^S} x_u^i \leq 1 \tag{4.1}$$



(a) Für den virtuellen Switch **(b)** Für den virtuellen Switch und die Server

Abbildung 4.10: Platzierungsvariablen für die Knotenplatzierungen (siehe auch Abb. 2.4)

Beispiel 4.7 (Platzierung eines virtuellen Switches)**.** Durch die Anwendung der TGG-Regel in Abb. 4.4b, wird für jede mögliche Platzierung eines virtuellen Switches i auf einem Substratknoten u ein Platzierungskandidat szk_u^i generiert. Danach wird für jeden Platzierungskandidaten szk_u^i eine Platzierungsvariable x_u^i erzeugt. In Abb. 4.10a werden die möglichen Platzierungen vsw1→ssw1, vsw1→ssr1 und vsw1→ssr2 als Platzierungskandidaten szk_{ssw1}^{vsw1}, szk_{ssr1}^{vsw1} und szk_{ssr2}^{vsw1} durch MT generiert. Danach werden die ILP-Variablen x_{ssw1}^{vsw1}, x_{ssr1}^{vsw1} und x_{ssr2}^{vsw1} aus den Platzierungskandidaten erstellt. Zur Sicherstellung, dass am Ende nur einer dieser Kandidaten szk_{ssw1}^{vsw1}, szk_{ssr1}^{vsw1} und szk_{ssr2}^{vsw1}, und damit auch nur einer der ILP-Variablen x_{ssw1}^{vsw1}, x_{ssr1}^{vsw1} und x_{ssr2}^{vsw1} ausgewählt wird, existiert die folgende ILP-Ungleichung:

$$x_{ssw1}^{vsw1} + x_{ssr1}^{vsw1} + x_{ssr2}^{vsw1} \leq 1$$

Platzierung eines virtuellen Servers Die Erstellung der ILP-Variablen x_u^i und der ILP-Ungleichungen für die Platzierung eines virtuellen Servers unter Anwendung der TGG-Regel in Abb. 4.8a erfolgt analog wie die Erstellung der ILP-Variablen und ILP-Ungleichungen für einen virtuellen Switch. Somit wird für jede mögliche Platzierung eines virtuellen Servers $i \in N^V (Sr_i = 1)$ auf einen

Substratserver $u \in N^S (Sr_u = 1)$ ein Platzierungskandidat szs_u^i durch die Anwendung der TGG-Regel erstellt (SzS→X). Danach werden die ILP-Variablen x_u^i auf Basis der Platzierungskandidaten szs_u^i erzeugt und durch die folgende Ungleichung sichergestellt, dass am Ende nur ein Platzierungskandidat szs_u^i für einen virtuellen Server i aus der Menge aller Kandidaten existiert.

$$\forall i \in N^V, Sr_i = 1 | \sum_{u=1}^{N^S, Sr_u=1} x_u^i \leq 1 \tag{4.2}$$

Beispiel 4.8 (Platzierung eines virtuellen Servers)**.** Die Erstellung der ILP-Variablen x_u^i für die möglichen Platzierungen der virtuellen Server i auf den Substratservern u wird analog zur Platzierung eines virtuellen Switches durchgeführt. Somit werden für alle Platzierungskandidaten szs_u^i eindeutige ILP-Variablen x_u^i erzeugt. In Abb. 4.10b sind diese ILP-Variablen x_{ssr1}^{vsr1}, x_{ssr2}^{vsr1}, x_{ssr1}^{vsr2} und x_{ssr2}^{vsr2} zusätzlich zu den ILP-Variablen für die Platzierung eines virtuellen Switches dargestellt. Damit jeder virtuelle Server i nur genau einmal platziert wird, werden die folgenden Ungleichungen generiert und zur ILP-Formulierung hinzugefügt.

$$x_{ssr1}^{vsr1} + x_{ssr2}^{vsr1} \leq 1$$
$$x_{ssr1}^{vsr2} + x_{ssr2}^{vsr2} \leq 1$$

Platzierung einer virtuellen Verbindung Die Generierung der ILP-Variablen für die Verbindungsplatzierungen $y_{p_{u,v}}^{l_{i,j}}$ einer virtuellen Verbindung $l_{i,j} \in L^V$ auf einen Substratpfad $p_{u,v} \in P^S$ wird analog zur Erstellung der ILP-Variablen für die Knotenplatzierungen x_u^i durchgeführt. Somit wird für jeden Platzierungskandidaten $vzp_{p_{u,v}}^{l_{i,j}}$ einer virtuellen Verbindung $l_{i,j}$ auf einen Substratpfad $p_{u,v}$ (VzP→Y), der durch die Anwendung der TGG-Regeln Abb. 4.9a und 4.9b erzeugt wird, eine ILP-Variable $y_{p_{u,v}}^{l_{i,j}}$ generiert. Auch hier muss sichergestellt werden, dass jede virtuelle Verbindung nur genau einmal auf einen Substratpfad platziert wird.

$$\forall i, j \in N^V | \sum_{u=1}^{N^S} \sum_{v=1}^{N^S} y_{p_{u,v}}^{l_{i,j}} \leq 1 \tag{4.3}$$

4.3.2.2 *ILP-Formulierung auf Basis der COCL-Zusicherungen*

Nachdem die ILP-Variablen für Knoten- und Verbindungsplatzierungen x_u^i und $y_{p_{u,v}}^{l_{i,j}}$ basierend auf den Platzierungskandidaten und die notwendigen ILP-Ungleichungen zur Einhaltung der TGG-Spezifikation generiert wurden, müssen nun noch die COCL-Zusicherungen in die ILP-Formulierung integriert werden, da diese Zusicherungen nicht in der MT-Konfiguration enthalten sind. Die COCL-Zusicherungen für das hier vorgestellte VNE-Problem haben hierbei folgende Struktur:

$$\begin{aligned} &\text{self.col.iterate(elem : T; sum : Integer = 0 | elem.a + sum)} \leq \text{ self.a} \\ &\text{mit col : Collection(T), ein Attribut a z.B. cpu, ram, hdd, bandbreite} \end{aligned} \tag{4.4}$$

Diese Zusicherungen können anhand der folgenden Form semantikerhaltend in ILP-Ungleichungen transformiert werden:

$$\sum_{e \in c} A_e x_s^e \leq A_s \tag{4.5}$$

Diese Transformation muss für die COCL-Zusicherungen C_{cpu}, C_{ram}, C_{hdd} und C_{bb} durchgeführt werden, welche sicherstellen, dass die Ressourcen der Substratelemente nicht überbucht werden. Zur Laufzeit werden somit die ILP-Ungleichungen ILP_{cpu}, ILP_{ram}, ILP_{hdd} und ILP_{bb} generiert und in die ILP-Formulierung integriert.

Beispiel 4.9 (ILP-Formulierungen für die COCL-Zusicherungen)**.** Die folgenden ILP-Ungleichungen für die COCL-Zusicherung C_{cpu} aus Abb. 4.10b werden generiert:

$$2x_{ssr1}^{vsr1} + 3x_{ssr1}^{vsr2} \leq 2$$
$$2x_{ssr2}^{vsr1} + 3x_{ssr2}^{vsr2} \leq 5$$

4.3.2.3 *Erstellung der Zielfunktion*

Die Optimierungsfunktion für das VNE-Problem wird anhand der in OCL definierten Zielfunktion (siehe Gleichung 3.6) abgeleitet. Dazu iterieren wir über alle virtuellen Links l und alle Substratpfade p und addieren alle Kosten basierend auf der Kostenmatrix *kosten(l,p)* auf. Da alle ILP-Variablen y_p^l basierend auf den Platzierungskandidaten vzp_p^l erzeugt wurden, können die ILP-Variablen direkt mit der Kostenfunktion *kosten(l,p)* multipliziert werden, wodurch die jeweiligen Kosten nur dann in die Summe mit einfließen, wenn eine ILP-Variable existiert und ausgewählt wurde. Dies resultiert in der folgenden ILP-Formulierung:

$$\text{min:} \sum_{l \in L^V} \sum_{p \in P^S} y_p^l \mathit{kosten(l,p)} \tag{4.6}$$

Die Invariante (*inv*) der Zielfunktion für die Verbindungen wurde bereits durch Gleichung 4.1, für die Platzierung eines virtuellen Switches, durch Gleichung 4.2, für die Platzierung eines virtuellen Servers und durch Gleichung 4.3, der Platzierung einer virtuellen Verbindung, sichergestellt.

4.3.3 *Beweis der Korrektheit und Optimalität*

In diesem Abschnitt zeigen wir auf, dass die Anwendung der vorgestellten Konstruktionsmethodik in einer MdVNE-Konfiguration resultiert, die korrekte und optimale Lösungen für das VNE-Problem basierend auf der modellbasierten Problemdefinition erzeugt. Dazu stellen wir in Definition 4.1 die durch das Metamodell charakterisierten Mengen, die Sprache und die Menge an OCL-Zusicherungen in Definition 4.2, die Mengen und die Sprache der Konstruktionsmethodik für MdVNE in Definition 4.3 und die Mengen in Bezug auf die Optimierungsfunktion in Definition 4.4 vor. Danach beweisen wir in Hauptsatz 4.1, dass die Menge

von korrekten und optimalen Lösungen der modellbasierten Problemdefinition identisch ist zur Menge an Lösungen die der MdVNE-Ansatz unter Verwendung der vorgestellten Konstruktionsmethodik besitzt (nach Schritt ②).

Definition 4.1 (Sprache des Metamodells)**.** Für ein Metamodell MM ist die Sprache $\mathfrak{L}(MM)$ die Menge aller gültigen und mit MM konformen Modelle.

Definition 4.2 (Sprache aller Modelle mit erfüllten Zusicherungen)**.** In dieser Definition charakterisieren wir Untermengen von $\mathfrak{L}(MM)$ danach, ob die Modelle in diesen Untermengen bestimmte Arten von OCL-Zusicherungen erfüllen. Die Sprache $\mathfrak{L}(C_{\text{OCL}})$ ist hierbei die Menge von allen Modellen in $\mathfrak{L}(MM)$, die die in der modellbasierten Problemdefinition definierten Menge von OCL-Zusicherungen C_{OCL} erfüllen: $\mathfrak{L}(C_{\text{OCL}}) := \{m \in \mathfrak{L}(MM) \mid m \models C_{\text{OCL}}\}$, wobei $m \models C_{\text{OCL}}$ eine erfüllbare Zielrelation darstellt, die fordert, dass m alle Zusicherungen aus C_{OCL} erfüllt. Die Menge $\mathfrak{L}(C_{\text{OCL}})$ enthält dabei alle Modelle, die sowohl die EOCL- als auch die COCL-Zusicherungen erfüllen mit $C_{\text{EOCL}} = C_{\text{OCL}} \cap \mathfrak{L}(EOCL)$, $C_{\text{COCL}} = C_{\text{OCL}} \cap \mathfrak{L}(COCL)$ und $C_{\text{OCL}} \subseteq \mathfrak{L}(OCL)$ Die Sprachen $\mathfrak{L}(OCL)$, $\mathfrak{L}(EOCL)$ und $\mathfrak{L}(COCL)$ repräsentieren Mengen von Zusicherungen, die durch OCL, EOCL oder COCL ausgedrückt werden können. Dabei gilt folgendes Beziehung: $\mathfrak{L}(OCL) = \mathfrak{L}(EOCL) \cup \mathfrak{L}(COCL)$. Die Menge $C_{\text{ROCL}} \subseteq \mathfrak{L}(EOCL)$ stellt eine Menge von relaxierten Zusicherungen aus der Menge der COCL-Zusicherungen $C_{\text{COCL}} \subseteq \mathfrak{L}(COCL)$ dar. Die Menge von Zusicherungen C_{ROCL} relaxiert die Menge von Zusicherungen C_{COCL} wenn die folgende Bedingung erfüllt ist: $\{\forall m \in \mathfrak{L}(MM) \mid m \models C_{\text{COCL}} \Rightarrow m \models C_{\text{ROCL}}\}$. Daher enthält $\mathfrak{L}(C_{\text{ROCL}})$, eine Obermenge von $\mathfrak{L}(C_{\text{COCL}})$, alle Modelle, die die relaxierten COCL-Zusicherungen erfüllen.

Definition 4.3 (Sprache aller aus einem Regelsatz generierten Modelle)**.** Wir definieren die Sprache $\mathfrak{L}(R_{MM})$ als Sprache für den Regelsatz R_{MM} für alle Modelle in $\mathfrak{L}(MM)$. Das ist die Menge aller Modelle, die ausgehend von dem leeren Modell durch eine endliche Anzahl von Regelanwendungen generiert werden kann. Dabei ist R_{MM} eine Menge von Regeln konstruiert durch die von Kehrer et al. [53] eingeführte Methodik, sodass $\mathfrak{L}(R_{MM}) = \mathfrak{L}(MM)$ (d.h. die Sprache, die durch den konstruierten Regelsatz R_{MM} beschrieben wird, erzeugt alle zum Metamodell MM konformen Modellinstanzen).
R_{EOCL} ist eine Menge von Regeln aus $\mathfrak{L}(R_{EOCL})$, die aus der Kombination von R_{MM} mit der Menge C_{EOCL} aus $\mathfrak{L}(C_{\text{EOCL}})$ von den EOCL-Zusicherungen abgeleitet wurde. Der Ableitungsprozess reichert hierbei den gegebenen Regelsatz R_{MM} mit Anwendungsbedingungen unter Verwendung der in [47, 80] eingeführten Methodik an. Analog hierzu fügen wir die Menge von relaxierten Zusicherungen C_{ROCL} zur Menge von C_{EOCL}-Zusicherungen hinzu ($\mathfrak{L}(R_{EOCL+ROCL})$), um daraus den Regelsatz $R_{EOCL+ROCL}$ abzuleiten ($\mathfrak{L}(C_{\text{EOCL}} \cup C_{\text{ROCL}})$).

Definition 4.4 (Optimierungsfunktion)**.** Die Menge $\text{opt}(C_{\text{COCL}}, \text{fg})$ enthält alle optimalen Lösungen für die gegebene Menge an Modellen, welche die Menge von C_{COCL}-Zusicherungen und die Optimierungsfunktion *fg* erfüllen. Die Menge $\mathcal{O}' := \text{opt}(C_{\text{COCL}}, \text{fg})$ für $\mathfrak{L}(R_{EOCL+ROCL})$ sind die durch MdVNE erzeugten optimalen Lösungen. Die Menge der optimalen Lösungen für die modellbasierte Problemdefinition ist hierbei $\mathcal{O} := \text{opt}(C_{\text{OCL}}, \text{fg})$.

Hilfssatz 4.1 (Für Definition 4.2)**.**
Behauptung:

$$\mathfrak{L}(C_{\text{OCL}}) = \mathfrak{L}(C_{\text{EOCL}}) \cap \mathfrak{L}(C_{\text{COCL}})$$

Beweis:

$$\begin{aligned}\mathfrak{L}(C_{\text{OCL}}) &= \mathfrak{L}(C_{\text{EOCL}} \cup C_{\text{COCL}}) \\ &= \{m \in \mathfrak{L}(MM) \mid m \models C_{\text{EOCL}} \wedge\ m \models C_{\text{COCL}}\} \\ &= \mathfrak{L}(C_{\text{EOCL}}) \cap \mathfrak{L}(C_{\text{COCL}})\end{aligned}$$

Hilfssatz 4.2 (Für Definition 4.3)**.**
Behauptung:

$$\mathfrak{L}(R_{EOCL+ROCL}) = \mathfrak{L}(C_{\text{EOCL}} \cup C_{\text{ROCL}})$$

Beweis:

$$\begin{aligned}\mathfrak{L}(R_{EOCL+ROCL}) &\overset{[47,\,80]}{=} \{m \in \mathfrak{L}(R_{MM}) \mid m \models C_{\text{EOCL}} \wedge m \models C_{\text{ROCL}}\} \\ &\overset{\text{Def. 4.2}}{=} \{m \in \mathfrak{L}(R_{MM}) \mid m \models C_{\text{EOCL}}\} \cap \{m \in \mathfrak{L}(R_{MM}) \mid m \models C_{\text{ROCL}}\} \\ &\overset{[47,\,80]}{=} \{m \in \mathfrak{L}(MM) \mid m \models C_{\text{EOCL}}\} \cap \{m \in \mathfrak{L}(MM) \mid m \models C_{\text{ROCL}}\} \\ &\overset{\text{Def. 4.2}}{=} \mathfrak{L}(C_{\text{EOCL}}) \cap \mathfrak{L}(C_{\text{ROCL}}) \\ &\overset{\text{Def. 4.2}}{=} \mathfrak{L}(C_{\text{EOCL}} \cup C_{\text{ROCL}})\end{aligned}$$

Im nachfolgenden beweisen wir Hauptsatz 4.1, welcher aussagt, dass der MdVNE-Ansatz bei Verwendung der modellbasierten Problemdefinition korrekte und optimale Lösungen nach der Kandidatenselektion ② liefert. Abbildung 4.11 skizziert die Idee des Beweises von Hauptsatz 4.1, der die Menge korrekter und optimaler Lösungen aus der modellbasierten Problemdefinition $\mathcal{O}$ auf der linken Seite und die Lösungen des MdVNE-Ansatzes unter Verwendung der vorgeschlagenen Konstruktionsmethode $\mathcal{O}'$ auf der rechten Seite liefert.

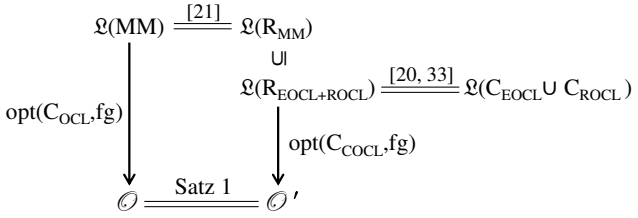


Abbildung 4.11: Schematische Darstellung für Hauptsatz 4.1

Hauptsatz 4.1 (Korrektheit und Optimalität der vorgestellten Konstruktionsmethodik)**.** Die Menge $\mathcal{O}'$, die die Lösungen von MdVNE (nach Schritt ② in Abb. 1.1) repräsentiert, ist gleich $\mathcal{O}$, der Menge der korrekten und optimalen Lösungen für die modellbasierte Problemdefinition. Um zu zeigen, dass $\mathcal{O} = \mathcal{O}'$ ist, muss die folgende Gleichheit beweisen werden (siehe Definitionen 4.2 und 4.3).

$$\text{opt}(\{m \in \mathfrak{L}(MM) \mid m \models C_{\text{OCL}}\}) = \text{opt}(\{m \in \mathfrak{L}(R_{EOCL+ROCL}) \mid m \models C_{\text{COCL}}\})$$

Beweis 4.1 (Korrektheit und Optimalität der vorgestellten Konstruktionsmethodik)**.** Wir beweisen Hauptsatz 4.1 durch Umformung der Mengen von $\mathcal{O}'$.

$$\begin{aligned}
\mathcal{O}' &= \text{opt}(\{m \in \mathfrak{L}(R_{EOCL+ROCL}) \mid m \models C_{\text{COCL}}\}) \\
&\overset{\text{Hilfssatz 4.2}}{=} \text{opt}(\{m \in \mathfrak{L}(MM) \mid m \models C_{\text{EOCL}} \wedge\ m \models C_{\text{ROCL}} \wedge\ m \models C_{\text{COCL}}\}) \\
&= \text{opt}(\{m \in \mathfrak{L}(MM) \mid m \models C_{\text{EOCL}} \wedge\ m \models C_{\text{COCL}}\}) \\
&\overset{\text{Hilfssatz 4.1}}{=} \text{opt}(\{m \in \mathfrak{L}(MM) \mid m \models C_{\text{OCL}}\}) \\
&= \mathcal{O}
\end{aligned}$$

Dies bedeutet, dass die Menge von optimalen Lösungen auf Basis der modellbasierten Problemdefinition gleich der Menge von Lösungen für den MdVNE-Ansatz unter Verwendung der vorgestellten Konstruktionsmethodik entspricht. □

4.3.3.1 *Korrektheit und Optimalität der ILP-Formulierung*

Die ILP-Formulierung aus dem Schritt der Kandidatenselektion ② wird anhand der Menge $\mathfrak{L}(R_{EOCL+ROCL})$ und den COCL-Zusicherungen (C_{COCL}) abgeleitet. Da in der Problemdefinition neben EOCL nur iterative Operationen mit einer Aggregation von einzelnen Elementen erlaubt sind (siehe Gleichung 3.1), können wir jede iterative Operation in C_{COCL} in eine ILP-Formulierung umwandeln. Auf diese Weise kann die Menge C_{COCL} als Summen konstanter Werte mit oder ohne ILP-Variablen dargestellt werden. Die ILP-Formulierung für $\mathfrak{L}(R_{EOCL+ROCL})$ wird durch eine generische Methodik zur Konsistenzprüfung unter Verwendung von TGGs [63] durchgeführt. Damit das implementierte MdVNE-Werkzeug korrekt funktioniert, müssen die folgenden Bedingungen erfüllt sein:

- Die Ableitung der ILP-Formulierungen aus den COCL-Zusicherungen C_{COCL} muss semantikerhaltend erfolgen.
- Der verwendete ILP-Löser muss korrekt arbeiten. Er muss die korrekten Lösungen berechnen und ihm müssen genügend Ressourcen (z.B. Rechenzeit) zur Verfügung gestellt werden, um eine optimale Lösung zu finden.
- Das verwendete MT-Werkzeug muss korrekt arbeiten. Das heißt, es muss alle Platzierungskandidaten, unter Berücksichtigung der EOCL- (C_{EOCL}) und

ROCL-Zusicherungen (C_{ROCL}), finden. Darüber hinaus müssen ihm alle notwendigen Ressourcen (z.B. Speicher und Rechenzeit) zur Verfügung gestellt werden, um diese Kandidaten zu finden.

- Die Erzeugung der Regeln aus dem Metamodell muss korrekt realisiert werden.
- Die Relaxationen müssen korrekt erzeugt sein und die Zusicherungen relaxieren.
- Die Anreicherung der TGG-Regeln um die Vorbedingungen muss korrekt und vollumfänglich durchgeführt werden.

5

UMSETZUNG FÜR STATISCHE UND DYNAMISCHE ANWENDUNGSSZENARIEN

Nachdem in den vorherigen Kapiteln anhand einer modellbasierten Problemdefinition eine MT- und ILP-Konfiguration mithilfe einer per-Konstruktion-korrekten Methodik erstellt wurde, wird in diesem Kapitel näher auf die Umsetzung, Codegenerierung und die Technologien zur Lösung von konkreten VNE-Problemen eingegangen. Zusätzlich werden neben statischen Szenarien, die durch MdVNE berechnet werden können, auch dynamische Szenarien betrachtet und es wird eine Erweiterung von MdVNE präsentiert, welche *inkrementelle* MT- und ILP-Technologien nutzt. Dazu stellen wir zuerst die grundlegenden Technologien für die Umsetzung und Codegenerierung vor und fokussieren uns danach auf die beiden Schritte Kandidatengenerierung ① und Kandidatenselektion ② aus Abb. 1.1. Dabei betrachten wir sowohl den statischen als auch den dynamischen Anwendungsfall getrennt voneinander. Zum Schluss diskutieren wir noch den Übergang von einem statischen hin zu einem dynamischen Anwendungsszenario und welcher Mehraufwand sich bei der Entwicklung für modellbasierte bzw. handgeschriebene Lösungsstrategien ergibt.

Somit stellt dieses Kapitel den wissenschaftlichen Beitrag **WB 4** und **WB 5** dar und basiert auf der Veröffentlichung [99].

5.1 Grundlegende Architektur

Zur Lösung von konkreten VNE-Problemen mithilfe von MdVNE kommen Technologien aus dem MT- und ILP-Bereich zum Einsatz. Diese Technologien erlauben es, dass aus einer deklarativen MT-Konfiguration (z.B. einem TGG-Regelsatz) ausführbarer Code generiert oder ILP-Formulierungen über Schnittstellen an ILP-Löser übergeben werden können.

Generell werden wir hier die beiden Fälle für statische und dynamische Szenarien unterscheiden, da diese Szenarien als Anwendungsszenarien in der Literatur üblich sind. Der Unterschied zwischen diesen beiden Szenarien besteht darin, dass im statischen Fall nur neue virtuelle Netzwerke eingebettet werden, vorhandene Netzwerke aber weder verändert, noch gelöscht oder migriert werden dürfen. Im dynamischen Fall hingegen ist es erlaubt, virtuelle Netzwerke zu erstellen, zu verändern, zu löschen oder zu migrieren, wobei in vielen Veröffentlichungen die Menge an erlaubten Möglichkeiten eingeschränkt wird (siehe Tabellen 2.4 und 2.5).

Zur Umsetzung von statischen Szenarien haben sich batchbasierte Ansätze etabliert, die jedes virtuelle Netzwerk oder mehrere Netzwerke gleichzeitig nach-

einander einbetten. Diese Vorgehensweise nutzen wir auch bei MdVNE. Somit ist es möglich, dass beispielsweise je nach Priorität oder Eintreffen des virtuellen Netzwerkes dieses auch bevorzugt vor anderen Netzwerken eingebettet wird.

Für dynamische Szenarien verwenden wir einen inkrementellen Ansatz, um nur die (meist kleinen) Modelländerungen im Vergleich zum gesamten Modell zu verarbeiten. So sind sowohl neue virtuelle Netzwerke, als auch Änderungen an einzelnen Servern im Vergleich zum Gesamtmodell (z.B. ein Rechenzentrum) als gering zu bezeichnen, wodurch vorherberechnete (Teil-)Ergebnisse oftmals wiederverwendet werden können. Dazu werden wir eine Erweiterung von MdVNE, den inkrementellen MdVNE-Ansatz (kurz iMdVNE), vorstellen und näher erläutern.

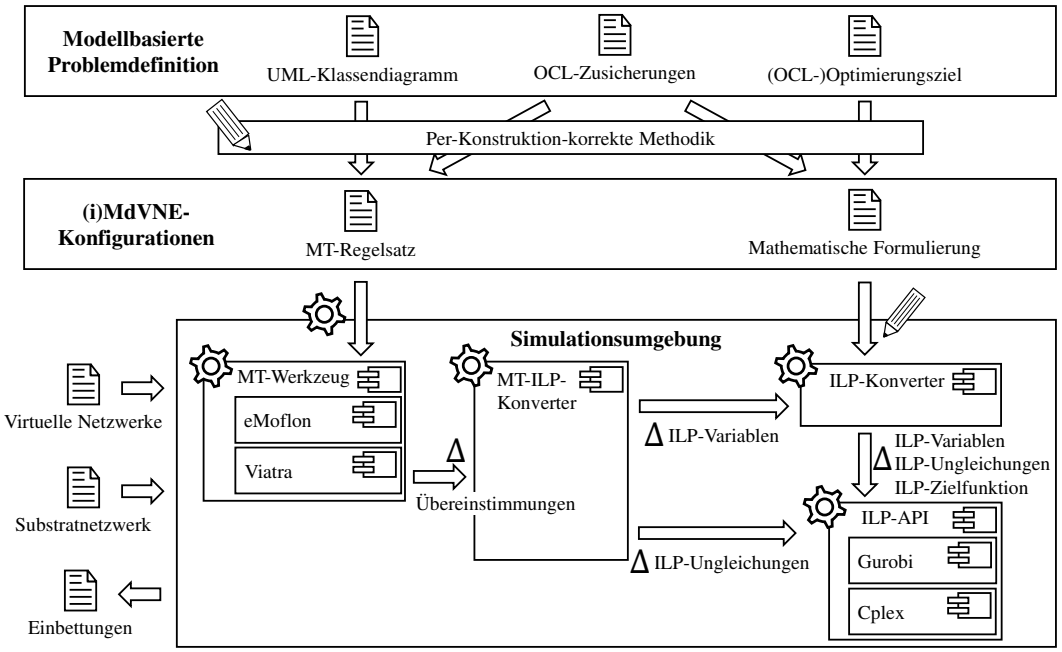


Abbildung 5.1: Überblick über die Umsetzung von (i)MdVNE

In Abb. 5.1 ist eine Übersicht über die Transformation der modellbasierten Problemdefinition in ausführbaren Code dargestellt, um (i)MdVNE zur Lösung von konkreten VNE-Problemen einzusetzen. Im ersten Schritt muss dazu eine modellbasierte Problemdefinition erstellt werden, welche sich aus einem UML-Klassendiagramm, OCL-Zusicherungen und einem Optimierungsziel, ebenfalls codiert in OCL, zusammensetzt. Diese Problemdefinition kann nun mit der vorgestellten per-Konstruktion-korrekten Methodik in eine (i)MdVNE-Konfiguration transformiert werden. Dieser Schritt wird aktuell manuell durchgeführt, wobei vielversprechende Ansätze zur (teilweisen) Automatisierung in der Literatur vorhanden sind und im Ausblick dieser Arbeit diskutiert werden. Der entstandene MT-Regelsatz kann durch ein MT-Werkzeug (z.B. eMoflon [62] oder Viatra [100]) direkt und automatisiert in ausführbaren Quellcode überführt werden. Zur Laufzeit werden diese MT-Regeln ausgeführt und es werden mithilfe eines MT-ILP-Konverters aus den gefundenen oder veränderten Übereinstimmungen (Deltas, **Δ**) automatisiert ILP-Variablen und ILP-Ungleichungen abgeleitet bzw. an-

gepasst. Für MdVNE kann dazu ein schon vorhandener und generischer MT-ILP-Konverter aus eMoflon verwendet werden (siehe [63]). Nachdem nun aus den (MT-)Übereinstimmungen die notwendigen ILP-Variablen und -Ungleichungen abgeleitet wurden, müssen die zusätzlichen ILP-Variablen, -Ungleichungen und die -Zielfunktion anhand der mathematischen Formulierung erstellt werden. Dazu wird im ersten Schritt ein ILP-Konverter genutzt, um anhand der Änderungen am Modell (Deltas, $\mathbf{\Delta}$) die (vorhandenen) ILP-Variablen, -Ungleichungen und die -Zielfunktion anzupassen. Die Konfiguration dieses ILP-Konverters anhand der mathematischen Formulierung muss für die Realisierung eines neuen VNE-Algorithmus aktuell noch manuell durchgeführt werden, wobei auch hier vielversprechende Ansätze zu einer (teilweisen) Automatisierung in der Literatur vorhanden sind und ebenfalls im Ausblick dieser Arbeit diskutiert werden. Durch das Wissen über die durchgeführten Modelländerungen (z.B. das Hinzufügen eines neuen virtuellen Netzwerkes, das Löschen eines Substratservers oder das Ändern einer virtuellen Ressource), der neuen und veränderten ILP-Variablen aus dem MT-ILP-Konverter und den ILP-Variablen und -Ungleichungen aus vorhergehenden Iterationen, kann der ILP-Konverter nun die finalen ILP-Variablen und -Ungleichungen zur Lösung des VNE-Problems ableiten. Nachdem die finalen ILP-Variablen bekannt sind, kann der ILP-Konverter auch die ILP-Zielfunktion generieren bzw. aktualisieren. Zusammen mit den ILP-Ungleichungen aus dem MT-ILP-Konverter werden diese Daten an eine generische ILP-API (z.B. Cardygan-ILP-API [86]) übergeben. Diese generische ILP-API realisiert nun die Anbindung an verschiedene ILP-Löser (z.B. Gurobi [42] oder Cplex [15]), wodurch das ILP-Problem gelöst und das Ergebnis interpretiert und weiterverwendet werden kann. Zusammen mit den virtuellen Netzwerken und dem Substratnetzwerk kann somit in einer Simulationsumgebung ein konkretes VNE-Problem gelöst, die gefundenen Einbettungen in das Modell übertragen und damit eine Evaluation dieses VNE-Algorithmus durchgeführt werden.

5.2 Kandidatengenerierung

Die Generierung der möglichen Platzierungskandidaten erfolgt anhand der deklarativen MT-Konfiguration, welche aus der modellbasierten Problemdefinition abgeleitet wird. Hierbei kann durch die MT-Konfiguration die Menge von allen möglichen Platzierungskandidaten auch weiter eingeschränkt werden, um beispielsweise die Zeit zum Lösen des VNE-Problems zu verringern oder heuristische Ansätze umzusetzen. Die Umsetzung dieser deklarativen Konfiguration in Java-Code wird durch eMoflon bzw. Viatra, generische und etablierte Wehrzeuge, durchgeführt, wodurch sichergestellt wird, dass (i) die Spezifikation der deklarativen Sprache (z.B. der TGG-Spezifikation) eingehalten wird und (ii) der erzeugte Quellcode durch die verbreitete Nutzung und intensive Weiterentwicklung annähernd fehlerfrei ist. Zusätzlich werden der Java-Laufzeitumgebung genügend Ressourcen (z.B. Speicher und Rechenzeit) zur Verfügung gestellt, damit alle korrekten Platzierungskandidaten gefunden werden können.

5.2.1 *Statisches Anwendungsszenario*

Zur Lösung von statischen VNE-Problemen verwenden wir den in Abschnitt 4.1 vorgestellten MdVNE-Ansatz. Er nutzt hierbei eine First In-First Out-Warteschlange, um die Reihenfolge der virtuellen Netzwerke festzulegen, wobei nach und nach entweder nur ein virtuelles Netzwerk oder mehrere Netzwerke gleichzeitig bearbeitet und eingebettet werden können. Somit stellt er einen batchbasierten Ansatz dar.

Anhand des TGG-Regelsatzes aus der MT-Konfiguration leitet eMoflon den ausführbaren Java-Quellcode ab. Zur Laufzeit wird nun anhand der TGG-Regeln aus Abb. 4.3 zuerst das Rechenzentrum (Substratnetzwerk) aufgebaut. Danach wird für das aktuell zu bearbeitende virtuelle Netzwerk der verfeinerte TGG-Regelsatz aus Abb. 4.4, 4.8 und 4.9 angewendet. Hierdurch werden alle Platzierungskandidaten `NzN`, `SzK`, `SzS` und `VzP` erzeugt, welche an den Schritt zur Kandidatenselektion ② weitergegeben werden.

Dabei wird für jede Iteration (Bearbeitung eines virtuellen Netzwerkes) das komplette Modell mit dem Substratnetzwerk, den eingebetteten virtuellen Netzwerken und deren Einbettungen neu generiert, da jede Iteration unabhängig von den internen Zuständen der vorherigen Iterationen ist. Die Iteration beginnt somit mit einem leeren Modell und führt alle TGG-Regeln für die virtuellen und Substratnetzwerke erneut aus.

5.2.2 *Dynamisches Anwendungsszenario*

Die Umsetzung von dynamischen Anwendungsszenarien wird in dieser Arbeit durch eine Erweiterung von MdVNE realisiert. Dieser inkrementelle modellbasierte Ansatz (kurz iMdVNE) unterstützt die Erstellung, Änderung und das Löschen sowohl von virtuellen Elementen, Netzwerken, deren Einbettungen oder den Substratelementen. Hierbei wird weiterhin die Korrektheit in Bezug auf die Anforderungen und die Optimalität in Bezug auf die Zielfunktion der gefundenen Lösungen sichergestellt. In Abb. 5.2 ist eine schematische Darstellung des iMdVNE-Ansatzes abgebildet, welcher wie MdVNE aus den beiden Schritten Kandidatengenerierung ① und Kandidatenselektion ② besteht. Da zur Umsetzung von dynamischen Anwendungsszenarien inkrementelle MT- und ILP-Technologien zum Einsatz kommen, werden nach der ersten Initialisierung nur noch die jeweiligen Modelländerungen (in Abb. 5.2 durch $\boldsymbol{\Delta}$ angedeutet) betrachtet. Durch die Verwendung von inkrementellen Technologien können diese Änderungen direkt bearbeitet und vorher berechnete Ergebnisse erneut wiederverwendet werden. Somit müssen für jede Iteration (Änderung) nicht alle Informationen von allen Netzwerken und Einbettungen erneut erzeugt und bearbeitet werden, sondern nur die Auswirkungen der jeweiligen Änderung.

Beim Übergang von einem statischen zu einem dynamischen Anwendungsszenario muss die Migration von bestehenden Einbettungen, die Wiedereinbettung von zuvor abgelehnten virtuellen Netzwerken und die Ablehnung eines bereits eingebetteten virtuellen Netzwerks mitberücksichtigt werden. Somit unterscheiden wir die folgenden drei Fälle: (i) *neue Einbettungen*, (ii) *Löschen von Einbettungen*

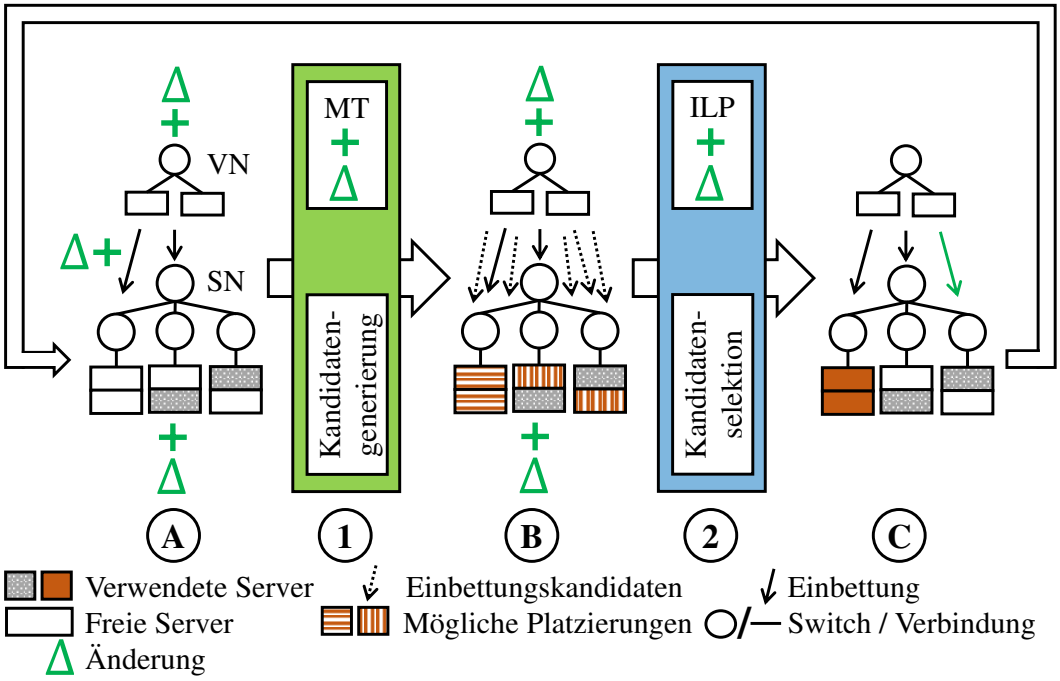


Abbildung 5.2: Schematische Darstellung von iMdVNE

und (iii) *Migration von Einbettungen*. Die ersten beiden Fälle (i) und (ii) können nur durch die Erstellung oder das Löschen von Elementen ausgelöst werden, wobei weitere Migrationen notwendig sein können, um optimale und korrekte Lösungen zu gewährleisten. Die Migration von bestehenden Einbettungen (iii) kann durch jedes Ereignis (Erstellung, Änderung oder Löschen) ausgelöst werden. Dabei unterscheiden wir zwei Migrationsstrategien: (a) *erforderliche Migration* und (b) *optimierende Migration*. Die erforderliche Migration (a) muss immer dann durchgeführt werden, wenn durch die Änderungen Zusicherungen verletzt werden. Somit stellt diese Migration sicher, dass die Einbettungen alle Zusicherungen respektieren. Die optimierende Migration (b) beinhaltet die erforderliche Migration, um sicherzustellen, dass alle Zusicherungen eingehalten werden, garantiert aber zusätzlich, dass alle Einbettungen optimal in Bezug auf die Zielfunktion sind. In Tabelle 5.1 werden die Ereignisse für das Erstellen, Ändern oder Löschen von Elementen bzw. deren Attributen wie Rechenkapazitäten (C), Arbeitsspeicher (R) und Festplattenspeicher (H) und die Auswirkungen auf das Erstellen (Neue E.) und das Löschen von Einbettungen (Lösch. E.) in den jeweiligen Migrationsstrategien (Erf. M., Opt. M.) dargestellt, wobei ein + andeutet, dass dieses Ereignis eine Anpassung notwendig machen kann. Die Änderung einer Eigenschaft repräsentiert hierbei beispielsweise das Ändern eines Standby-Servers.

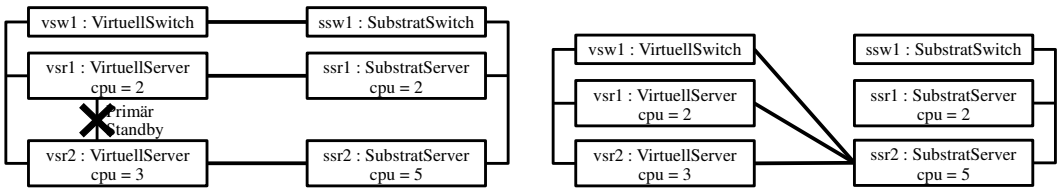
Beispiel 5.1 (Dynamisches Anwendungsszenario)**.** Nachdem das komplette virtuelle Netzwerk in das Substratnetzwerk eingebettet wurde (siehe Abb. 3.3a) wird nun die Verbindung zum Standby-Server *vsr2* entfernt (Änderung der Eigenschaft, siehe Abb. 5.3a). Da durch diese Änderung keine Zusicherungen verletzt werden, muss keine erforderliche Migration durchgeführt wer-

Tabelle 5.1: Migrationsstrategien für Änderungen

Modelländerung	Neue E.	Lösch. E.	Erf. M.	Opt. M.
Vir. Element erstellen	+	−	+	+
Sub. Element erstellen	−	−	−	+
Vir. Ressource verringern *C/R/H*	−	−	−	+
Vir. Ressource erhöhen *C/R/H*	−	−	+	+
Sub. Ressource verringern *C/R/H*	−	−	+	+
Sub. Ressource erhöhen *C/R/H*	−	−	−	+
Eigenschaft ändern	−	−	+	+
Vir. Element löschen	−	+	−	+
Sub. Element löschen	−	+	+	−

den. Allerdings könnte sich durch eine Migration des virtuellen Switches *vsw1* oder der virtuellen Server *vsr1* oder *vsr2* ein niedrigerer Wert für die Zielfunktion ergeben, da beispielsweise die Kosten für die Migration niedriger sind als der Gewinn durch die verringerten Kommunikationskosten. Durch eine optimierende Migration wird daher in diesem Beispiel der virtuelle Switch *vsw1* und der virtuelle Server *vsr1* auf den Substratserver *ssr2* verschoben (siehe Abb. 5.3b).

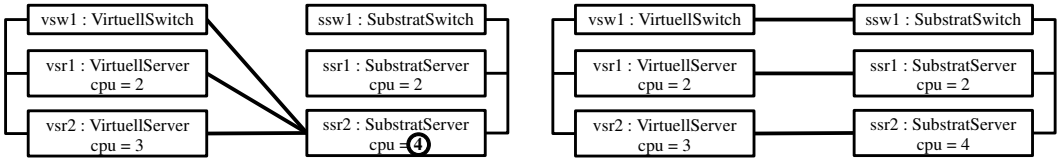
Danach wird die Rechenkapazität des Substratservers *ssr2* von 5 auf 4 CPU-Kernen reduziert (siehe Abb. 5.4a). Dadurch wird die Zusicherung verletzt, dass die Summe von virtuellen Ressourcen auf einem Substratelement, die Ressourcen dieses Elements nicht überschreiten dürfen, da die beiden virtuellen Server *vsr1* und *vsr2* zusammen 5 CPU-Kerne zum Betrieb benötigen. Somit muss in diesem Fall eine erforderliche Migration durchgeführt werden, da mindestens einer dieser beiden virtuellen Server migriert werden muss, um alle Zusicherungen wieder zu gewährleisten. In diesem Beispiel wird neben dem virtuellen Server *vsr1* auch noch der virtuelle Switch *vsw1* migriert, was in Abb. 5.4b dargestellt ist.



(a) Entfernen eine Standby-Verbindung **(b)** Neue Einbettungen nach dem Entfernen

Abbildung 5.3: Entfernen einer Standby-Verbindung nach Abb. 3.3a

Um nun im Schritt Kandidatengenerierung ① die Platzierungskandidaten zu erstellen, zu aktualisieren oder zu löschen, nutzen wir inkrementelle MT-Techniken [69]. Diese Techniken nutzen einen inkrementellen Mustervergleich, der alle Vorkommen eines Musters in einem Modell findet, welche als Übereinstimmungen bezeichnet werden. Diese Übereinstimmungen können als injektive Zuordnungen von Modellelementen zu den Elementen im Muster verstanden werden, sodass



(a) Änderung der Rechenkapazität von *ssr2* **(b)** Neue Einbettungen nach der Änderung

Abbildung 5.4: Änderung der Rechenkapazität eines Substratservers nach Abb. 5.3b

Tabelle 5.2: Auswirkung der Ereignisse auf die Übereinstimmungen im Netzwerk

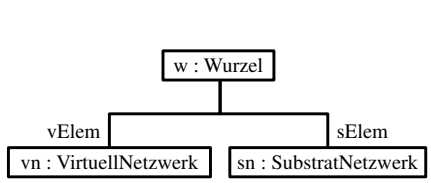
Ereignis	**Neue Ü.**	**Geänd. Ü.**	**Gelösch. Ü.**
Element erstellen	+	−	−
Vir. Ressource verringern *C/R/H*	−	+	+
Vir. Ressource erhöhen *C/R/H*	+	+	−
Sub. Ressource verringern *C/R/H*	−	+	+
Sub. Ressource erhöhen *C/R/H*	+	+	−
Eigenschaft ändern	+	−	+
Element löschen	−	−	+

jedes Modellelement maximal einmal in einer Übereinstimmung vorhanden sein kann. Der Vorteil der Verwendung des inkrementellen Mustervergleichs besteht darin, dass die Änderungen (Deltas) in einem Modell nachverfolgt werden und Benachrichtigungen über neu gefundene oder verschwundene Übereinstimmungen versendet werden können, anstatt alle Übereinstimmungen bei Änderungen von Grund auf neu zu sammeln. Da beim inkrementellen Mustervergleich alle Deltas überwacht werden, unterscheiden wir zwischen drei Fällen:

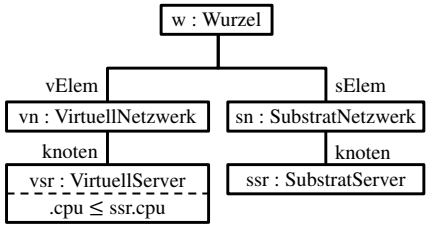
1. eine neue Übereinstimmung wird hinzugefügt (neue Ü.),
2. eine bestehende Übereinstimmung wird geändert (geänd. Ü.) oder
3. eine Übereinstimmung verschwindet (gelöschte Ü.).

In Tabelle 5.2 werden diese Fälle zusammen mit den möglichen Ereignissen für die Netzwerke und deren Einbettungen in iMdVNE dargestellt.

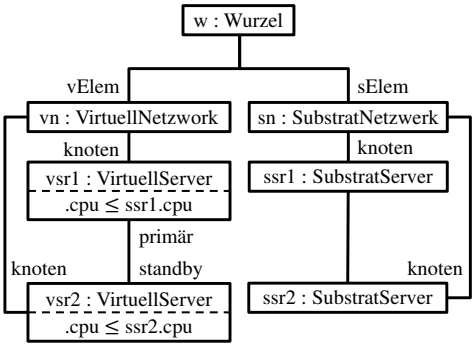
Eine Auswahl der bei iMdVNE eingesetzten deklarativen MT-Muster zum Erstellen, Ändern oder dem Löschen von Platzierungskandidaten ist in Abb. 5.5 dargestellt. Das Muster in Abb. 5.5a repräsentiert die Einbettung eines virtuellen Netzwerks in ein Substratnetzwerk. Die Variablen `w`, `vn` und `sn` repräsentieren Elemente im Modell, wobei `w` ein übergeordnetes Element darstellt, welches immer im Modell vorhanden ist und als Wurzel für das Muster dient. Eine Übereinstimmung für dieses Muster ist dann gegeben, wenn alle Variablen mit deren jeweiligen Assoziationen im Modell existieren. Für jede Übereinstimmung wird daraufhin ein neuer Platzierungskandidat `NzN` generiert, was in einer Menge von allen Kombinationen zwischen virtuellen und Substratnetzwerken resultiert. Konkret wird für jede Übereinstimmung eine Assoziation `VirtuellNetzwerk.substrat` und `SubstratNetzwerk.virtuell` zwischen den beiden Elementen von `vn` und



(a) MT-Muster für die Netzwerkeinbettungen



(b) MT-Muster für die Servereinbettungen



(c) MT-Muster für die Servereinbettungen mit einem Standby-Server

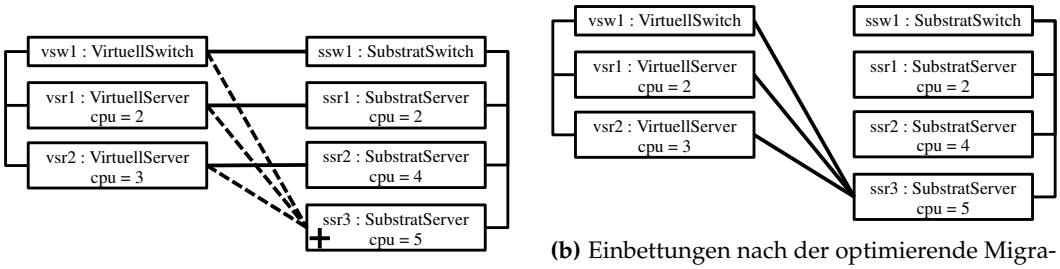
Abbildung 5.5: MT-Muster für die Einbettung des Netzwerks und der Server

sn erzeugt, welche einen neuen potenziellen Platzierungskandidaten repräsentieren. Dieses Muster dient auch dazu, Platzierungskandidaten zu finden und zu löschen, die durch die Erstellung oder das Löschen von Netzwerken betroffen sind. Falls ein Netzwerk gelöscht wird, verschwindet die Übereinstimmung für das Muster und alle Platzierungskandidaten (*VirtuellNetzwerk.substrat* und *SubstratNetzwerk.virtuell*) können gelöscht werden. In Abb. 5.5b wird das Muster zur Platzierung eines virtuellen Servers auf einen Substratserver dargestellt. Ähnlich zu Abb. 5.5a stellen die Variablen im Muster konkrete Elemente dar, wobei in diesem Fall zusätzlich geprüft wird, dass die Anzahl der CPU-Kerne im virtuellen Server die CPU-Kerne im Substratserver nicht überschreitet (*.cpu* $\leq$ *ssr.cpu*). Analog zu Abb. 5.5a wird durch das Finden einer neuen Übereinstimmung (z.B. ein neuer Server wird erstellt) ein neuer Platzierungskandidat in Form einer Assoziation zwischen *vsr* und *ssr* erzeugt (vsr→ssr). Wenn nun ein Server gelöscht oder die Anzahl der CPU-Kerne so geändert wird, dass die Bedingung *.cpu* $\leq$ *ssr.cpu* nicht mehr gilt, verschwindet diese Übereinstimmung. Dies führt dazu, dass Assoziationen zwischen *vsr* und *ssr* und deren Platzierungskandidaten vsr→ssr gelöscht werden. Das letzte Muster in Abb. 5.5c repräsentiert den Fall, dass *vsr2* als Standby-Server für *vsr1* agiert, wobei wieder sichergestellt wird, dass auf den jeweiligen Substratservern *ssr1* und *ssr2* genügend CPU-Kerne vorhanden sind. Falls eine Übereinstimmung für dieses Muster neu gefunden wird, muss nun sichergestellt werden, dass der primäre und Standby-Server nicht auf demselben Substratserver platziert werden. Beim Verschwinden einer Übereinstimmung für dieses Muster ist allerdings keine weitere Aktion notwendig, da weiterhin alle Zusicherungen eingehalten werden.

Beispiel 5.2 (Inkrementeller Mustervergleich)**.** Als Ausgangspunkt dient die Modellinstanz aus Abb. 3.3a. Nun wird die Verbindung zwischen dem primären Server *vsr1* und dem Standby-Server *vsr2* entfernt (siehe Abb. 5.3a). Durch das Entfernen dieser Assoziation verschwindet auch die dazu passende Übereinstimmung für das in Abb. 5.5c dargestellte Muster. Da durch das Verschwinden keine weiteren Zusicherungen verletzt werden, muss keine erforderliche Migration durchgeführt werden. Allerdings könnte eine optimierende Migration durchgeführt werden, falls dies konfiguriert wurde.

Als nächstes Ereignis wird die Anzahl der CPU-Kerne vom Substratserver *ssr2* von 5 auf 4 reduziert (siehe Abb. 5.4a), wobei die Zusicherung verletzt wird, dass die Ressourcen der Substratelemente nicht überbucht werden dürfen (siehe Anforderung K3). Die Verletzung dieser Zusicherung kann allerdings nur im Schritt zur Kandidatenselektion erkannt werden, da die Bedingung $.cpu \leq ssr.cpu$ aus Muster Abb. 5.5b weiterhin erfüllt ist.

Im nächsten Schritt wird ein neuer Substratserver *ssr3* mit 5 CPU-Kernen zum Substratnetzwerk hinzugefügt (siehe Abb. 5.6a). Durch dieses Ereignis werden neue Übereinstimmungen für das Muster aus Abb. 5.5b gefunden, wodurch die drei neuen Platzierungskandidaten vsw1→ssr3 (szk_{ssr3}^{vsw1}), vsr1→ssr3 (szs_{ssr3}^{vsr1}) und vsr2→ssr3 (szs_{ssr3}^{vsr2}) erzeugt werden. Bei diesem Ereignis müssen wir zwischen den beiden Migrationsstrategien unterscheiden. Im Fall der erforderlichen Migration werden die Einbettungen aus Abb. 5.4b beibehalten, da durch das Hinzufügen des neuen Substratservers *ssr3* keine Zusicherung verletzt wurde. Im Fall der optimierenden Migration ändern sich die Einbettungen, da eine Lösung mit einem niedrigeren Wert für die Zielfunktion vorhanden ist. Somit werden alle virtuellen Elemente *vsw1*, *ssr1* und *ssr2* auf dem neuen Substratserver *ssr3* platziert (siehe Abb. 5.6b).



(a) Hinzufügen eines neuen Substratservers

(b) Einbettungen nach der optimierende Migration

Abbildung 5.6: Hinzufügen eines neuen Substratservers nach Abb. 5.4b

5.3 Kandidatenselektion

Nachdem im Schritt Kandidatengenerierung ① alle möglichen Kandidaten durch die MT-Konfiguration erzeugt wurden, wird nun im Schritt Kandidatenselektion ② eine Lösung berechnet, welche garantiert alle Zusicherungen aus der modellbasierten Problemdefinition erfüllt und zusätzlich optimal in Bezug auf die Zielfunktion ist. Hierfür nutzen wir ILP-Technologien, da durch ILP diese Eigenschaf-

ten erfüllt sind und es hoch entwickelte und in der Praxis eingesetzte ILP-Löser gibt.

5.3.1 *Statisches Anwendungsszenario*

Die Generierung der ILP-Formulierungen für statische Anwendungsszenarien mithilfe von MdVNE wurde bereits in Abschnitt 4.3.2 vorgestellt. Da hierbei in jeder Iteration (jede neue Einbettung eines virtuellen Netzwerks) die ILP-Formulierung neu aufgebaut wird, werden alle ILP-Variablen und –Einschränkungen direkt aus der modellbasierten Problemdefinition und anhand der Platzierungskandidaten aus der Kandidatengenerierung erstellt. Zusätzlich werden noch ILP-Einschränkungen zum Einhalten der TGG-Spezifikation hinzugefügt und die Zielfunktion erzeugt, welche auf den bereits generierten ILP-Variablen aufbaut.

5.3.2 *Dynamisches Anwendungsszenario*

Im dynamischen Anwendungsszenario werden nur die inkrementellen Änderungen (Deltas) im Modell verarbeitet, wodurch zu Beginn des Szenarios eine initiale ILP-Formulierung erzeugt wird, welche durch jedes neue Ereignis (z.B. Löschen eines virtuellen Servers) aktualisiert wird. Dazu werden sowohl die Deltas (z.B. Löschen des virtuellen Servers) als auch die Änderungen, die durch die Kandidatengenerierung erzeugt wurden, integriert. Hierbei lassen sich vier Fälle bei der Aktualisierung der ILP-Formulierung unterscheiden:

1. Hinzufügen von neuen ILP-Variablen oder Einschränkungen,
2. Anpassung der Zielfunktion,
3. Aktualisierung bestehender Einschränkungen und
4. Löschen bestehender ILP-Variablen oder Einschränkungen.

Dabei muss beachtet werden, dass immer bekannt ist, welche ILP-Variable aus welchen Platzierungskandidaten erzeugt wurde, damit die Änderungen aus der Kandidatengenerierung auch in die ILP-Formulierung übertragen werden können. Durch dieses Vorgehen können je nach interner Umsetzung des ILP-Löser vorherberechnete (Zwischen-)Ergebnisse oder Approximationen (teilweise) wiederverwendet werden.

Ausgehend von den Ereignissen (z.B. Löschen eines virtuellen Servers) aus der Kandidatengenerierung werden entweder neue Übereinstimmungen für Platzierungskandidaten gefunden, bestehende Übereinstimmungen aktualisiert oder vorherige Übereinstimmungen entfernt. Durch das Hinzufügen von Servern oder Switchen werden dabei neue Übereinstimmungen für Platzierungskandidaten erzeugt, wodurch auch neue ILP-Variablen und –Einschränkungen abgeleitet werden müssen. Analog führt das Löschen von Elementen dazu, dass die betreffenden Übereinstimmungen verschwinden und somit auch die ILP-Variablen gelöscht und falls erforderlich die involvierten ILP-Einschränkungen angepasst werden müssen. Die Aktualisierung einer Übereinstimmung kann dabei Einfluss auf

Tabelle 5.3: Auswirkung der Ereignisse auf die bestehende ILP-Formulierung

Modelländerung	**Neue V./E.**	**Änd. Z.**	**Änd. E.**	**Lösch. V./E.**
Neue Übereinst.	+	+	+	–
Aktualisierung vir. Übereinst.	–	+	+	–
Aktualisierung sub. Übereinst.	–	–	+	–
Lösche Übereinst.	–	+	+	+

die ILP-Einschränkungen und/oder die –Zielfunktion besitzen. In Tabelle 5.3 sind diese verschiedenen Möglichkeiten nochmal dargestellt, wobei die Auswirkungen der Modelländerungen auf die Variablen (V), Einschränkungen (E) oder der Zielfunktion (Z) aufgezeigt werden.

Nachdem alle Änderungen für ILP-Variablen und -Ungleichungen durchgeführt wurden, muss die Zielfunktion inklusive der Migrationskosten angepasst werden. Dabei können in die Migrationskosten eine Vielzahl von Gegebenheiten und Kosten mit eingerechnet werden, wie beispielsweise die Anzahl an notwendigen Migrationen, die benötigte Zeit, die Menge der betreffenden Ressourcen, die Anzahl der Hops, der Energieverbrauch, die benötigten Bandbreiten oder die erwarteten Ausfallzeiten [33, 39]. In dieser Arbeit beschränken wir uns auf die Ressourcen der virtuellen Server die umgezogen werden müssen, da die Menge an Ressourcen und Daten Auswirkungen auf die benötigte Bandbreite oder die notwendige Zeit zum Umzug hat. Zur Berechnung der Migrationskosten müssen zunächst die beiden folgenden Fälle unterschieden werden:

(a) Der virtuelle Server ist seit der letzten Einbettung neu hinzugekommen und wurde bisher noch nicht eingebettet.

(b) Der virtuelle Server ist nach der vorhergehenden Einbettung platziert gewesen und ist nun nicht mehr eingebettet.

Somit ergibt sich bei der Betrachtung der Migrationskosten eine zeitliche Reihenfolge, da neben der aktuellen Einbettung (Zeitpunkt *t*, vor dem Ausführen des Algorithmus) auch die vorausgehende Einbettung (Zeitpunkt *t-1*) mit den dazwischen durchgeführten Ereignissen beachtet werden muss. Analog hierzu existieren die Platzierungsvariablen $x_u^i(t-1)$ und $x_u^i(t)$, welche jeweils die Einbettungen (Knotenplatzierungen) zum Zeitpunkt t-1 und t darstellen. Für Fall (a) existiert der virtuelle Server i zum Zeitpunkt t-1 noch nicht, wodurch auch keine Platzierung vorliegt. Dies kann durch $x_u^i(t-1) = 0$ ausgedrückt werden, wobei zur Laufzeit in der Simulationsumgebung die Variable $x_u^i(t-1)$ noch gar nicht existieren würde. Für den Fall (b) war der virtuelle Server i zum Zeitpunkt t-1 platziert ($x_u^i(t-1) = 1$) und ist nun vor dem Ausführen der jetzigen Einbettung (zum Zeitpunkt t) nicht mehr eingebettet. Dies kann durch verschiedene Ereignisse eingetreten sein (z.B. durch das Löschen eines Substratservers oder durch die Veränderung der virtuellen Ressourcen). Da hierdurch die Anforderung K2 (alle virtuellen Server müssen eingebettet sein) verletzt ist, muss nun sichergestellt werden, dass nach der Einbettung dieser virtuelle Server i erneut eingebettet ist.

Tabelle 5.4: Möglichkeiten für $x_u^i(t-1)$, $x_u^i(t)$ und m_u^i

$x_u^i(t-1)$	$x_u^i(t)$	m_u^i
0	0	0
0	1	0
1	0	1
1	1	0

Somit ergeben sich vier Fälle für die beiden Platzierungsvariablen $x_u^i(t-1)$ und $x_u^i(t)$, welche in Tabelle 5.4 zusammengefasst sind.

Zur Berechnung der Migrationskosten müssen nur die Fälle berücksichtigt werden, in welchem der virtuelle Server i seit der letzten Einbettung auf einen anderen Substratserver verschoben wurde. Daher muss zum Zeitpunkt t-1 der virtuelle Server i eingebettet sein und zum Zeitpunkt t nicht mehr. Um dies in die ILP-Zielfunktion zu integrieren, kann eine zusätzliche Variable $m_u^i \in \{0,1\}$ verwendet werden, welche genau dann eins ist, wenn eine Migration des virtuellen Servers i durchgeführt wird. Für die Berechnung der Kosten für die Migration wird in dieser Arbeit die Summe aller CPU-Kerne, des Arbeitsspeichers und des Festplattenspeichers des zu migrierenden virtuellen Servers herangezogen, da diese Ressourcen direkte Auswirkungen auf die Dauer und benötigte Bandbreite der Migration hat. Die ILP-Zielfunktion ILP_{Ziel} kann daher zu folgender Zielfunktion mit Migrationskosten erweitert werden:

$$\begin{aligned} \text{min: } & \sum_{p_{u,v} \in P^S} \sum_{l_{i,j} \in L^V} y_{p_{u,v}}^{l_{i,j}} \mathit{kosten}(l_{i,j}, p_{u,v}) + \sum_{i \in N^V} \sum_{u \in N^S} x_u^i m_u^i \mathit{migrationskosten}(\text{i}) \\ & \text{mit } \mathit{migrationskosten}(\text{i}) = \sum_i C_i^V + M_i^V + S_i^V \end{aligned} \tag{5.1}$$

Nachdem diese Änderungen in die ILP-Formulierung integriert wurden, kann der ILP-Löser eine Lösung für diese Problemstellung berechnen, welche dann in das Modell übertragen werden kann.

5.4 Übergang vom statischen zum dynamischen Szenario

Die Nutzung von MdVNE für statische und iMdVNE für dynamische Anwendungsszenarien erfordert zuerst eine modellbasierte Problemdefinition, die mithilfe der per-Konstruktion-korrekten Methodik manuell in einen MT-Regelsatz und eine mathematische Formulierung überführt wird (siehe Abb. 5.1). Die Generierung von Quellcode und die Anpassung an das MT-Werkzeug und den ILP-Löser erfolgt hierbei fast vollständig automatisiert, wodurch eine manuelle Überführung in Quellcode nur in sehr geringem Umfang notwendig ist. Dadurch verschiebt sich der Fokus bei der Entwicklung von neuen (i)MdVNE-basierten Lösungsstrategien im Vergleich zu handgeschriebenen Implementierungen von der Erstellung von ausführbarem Quellcode hin zu einer abstrakteren Spezifikationstätigkeit. Somit kann die Qualität, Wartbarkeit und Anpassbarkeit des (automatisch generierten) Quellcodes und der gefundenen Lösungen verbessert wer-

den. Die Unterschiede bei der Überführung der modellbasierten Problemdefinition und MdVNE-Konfiguration für statische Anwendungsszenarien hin zu dynamischen Szenarien ist hierbei sehr gering, da die grundlegende modellbasierte Problemdefinition bis auf das Hinzufügen einer Migrationskostenfunktion erhalten bleibt. Somit müssen zur Spezifikation eines dynamischen Szenarios lediglich die Migrationskosten spezifiziert und in die Zielfunktion integriert werden. Mithilfe dieser angepassten Problemdefinition (für das dynamische Szenario) und der per-Konstruktion-korrekten Methodik aus Kapitel 4 kann nun eine Konfiguration für iMdVNE abgeleitet werden, welche, analog zum statischen Szenario, garantiert korrekte und optimale Lösungen findet. Für die Erstellung der Konfiguration sind allerdings mehr Fälle und Iterationen notwendig, da durch die erhöhte Anzahl von Ereignissen (Erstellung, Änderung oder Löschen von Netzwerken, Elementen und Ressourcen) mehr (Teil-)Muster und Graph-Einschränkungen betrachtet werden müssen.

Für eine handgeschriebene ILP-basierte und mit (i)MdVNE vergleichbare Implementierung ist eine Überführung von einem statischen hin zu einem dynamischen Anwendungsszenario mit einem erheblichen Mehraufwand verbunden. So müssen bei der Überführung alle denkbaren Abfolgen von Ereignissen zum Erstellen, Ändern und Löschen von virtuellen und Substratnetzwerken, Elementen und Ressourcen unter Berücksichtigung der möglichen Netzwerktopologien betrachtet werden. Zusätzlich stellt der Beweis, dass die Lösungsstrategie immer korrekte und optimale Lösungen findet für diese enorme Anzahl an möglichen Abfolgen eine Herausforderung dar. Auch die Integration einer inkrementell operierenden Lösungsstrategie, wodurch sich der Rechenaufwand möglichst proportional zu der Größe der Änderung und nicht zur Größe des zugrundeliegenden Modells verhält, ist mit einem erhöhten Aufwand verbunden. Dies erfordert die manuelle Implementierung eines anpassbaren Generators zur Erstellung von ILP-Formulierungen mit einer internen (inkrementell agierenden) Buchhaltung über die verwendeten Variablen, Ungleichungen, deren Abhängigkeiten untereinander und deren Einfluss auf die Zielfunktion.

Zusammenfassend kann man sagen, dass die Überführung von statischen in dynamische Anwendungsszenarien für (i)MdVNE-basierten Lösungsstrategien mit einem sehr geringen Mehraufwand möglich ist. Dabei verschiebt sich der Fokus beim Entwicklungsaufwand von der Erstellung ausführbaren Quellcodes (Low-Level) hin zur Anpassung der Spezifikation (High-Level) mit einem erwarteten Zugewinn an Codequalität und der Qualität der Lösungen (korrekt und optimal). Trotz dieser Fokusverschiebung bleibt die Recheneffizienz der (i)MdVNE-basierten Implementierungen vergleichbar zu einer handgeschriebenen und -optimierten Implementierung und ist durch die Integration von inkrementellen Technologien teilweise sogar merklich effizienter, wie es die Evaluation zeigen wird.

6

EVALUATION

In diesem Kapitel untersuchen wir in einer experimentellen Evaluation die zuvor vorgestellten modellbasierten Lösungsstrategien für die Einbettung von virtuellen Netzwerken in Rechenzentren. Dazu evaluieren wir die beiden folgenden Anwendungsszenarien:

- *Statisches Anwendungsszenario*: Dieses wird mithilfe von MdVNE gelöst.
- *Dynamisches Anwendungsszenario*: Dieses wird mithilfe von iMdVNE gelöst.

Die Ziele dieser Evaluation können in die beiden folgenden Kategorien unterteilt werden:

A *Untersuchung der Effizienz* bezüglich der Laufzeit und der Skalierbarkeit.

B *Untersuchung der Effektivität* in Bezug auf die Korrektheit und Optimalität.

Das Kapitel beginnt mit einer Vorstellung des Versuchsaufbaus für beide Anwendungsszenarien in Abschnitt 6.1. Danach gehen wir auf die beiden Anwendungsszenarien (statisch und dynamisch) getrennt voneinander ein. Für jedes Szenario werden zuerst Forschungsfragen festgelegt, die in den darauffolgenden Abschnitten diskutiert und beantwortet werden. Nach einer Diskussion bezüglich der Validität der Ergebnisse folgt eine Zusammenfassung der Evaluation des jeweiligen Szenarios. Wir beginnen zuerst mit dem statischen Anwendungsszenario in Abschnitt 6.2, gefolgt von dem dynamischen Szenario in Abschnitt 6.3. Zum Abschluss vergleichen wir noch in Abschnitt 6.4 die Ergebnisse von MdVNE und iMdVNE.

6.1 Versuchsaufbau

Für die experimentelle Evaluation simulieren wir die Einbettung von virtuellen Netzwerken in ein Rechenzentrum sowohl für den statischen als auch für den dynamischen Fall anhand der Anforderungen aus Abschnitt 2.2. Dabei wird zusätzlich die anwendungsfallspezifische Anforderung mit integriert, dass ein virtueller Server auch ein Hochleistungsserver sein kann. Dieser Hochleistungsserver darf dabei nur auf einem Hochleistungsserver im Substratnetzwerk betrieben werden. Zusätzlich dürfen auf diesen Substratservern auch nur virtuelle Server betrieben werden, die diese Funktionalität benötigen.

Das Rechenzentrum (Substratnetwerk) ist als 2-Tier Netzwerk (siehe Abb. 2.1a) mit zwei Kernswitchen modelliert, die mit den Rackswitchen über eine Bandbreite von 10 GB/s verbunden sind. Jedes Rack besteht dabei aus 10 Servern mit

jeweils 32 Prozessorkernen (*cpu*), 512 GB Arbeitsspeicher (*ram*) und 1 TB Festplattenspeicher (*hdd*). Dabei sind pro Rack zwei Hochleistungsserver vorhanden, auf welchem nur virtuelle Server betrieben werden dürfen, die ebenfalls Hochleistungsserver sind. Alle Server sind mit dem jeweiligen Rackswitch über eine Verbindung mit der Bandbreite von 1 GB/s verbunden. Für die Substratpfade werden alle möglichen, zyklusfreien Pfade mit einer Länge von maximal vier Hops generiert, da dies die maximal notwendige Pfadlänge ist, um einen Server von Rack A mit einem Server von Rack B zu verbinden. In dieser Evaluation werden zwei Substratnetzwerkgrößen untersucht: (i) ein kleines Netzwerk (kSN) mit 8 Racks (insgesamt 80 Server) und (ii) ein großes Netzwerk (gSN) mit 12 Racks (insgesamt 120 Server).

Die virtuellen Netzwerke werden als Sterntopologie mit einem zentralen Switch und 2 bis 10 Servern modelliert (analog zu [109]). Die Ressourcen der virtuellen Server sind aus den realistischen Daten des Bitbrain-Datensatzes [91] abgeleitet. Die Anzahl der Prozesskerne liegt somit zwischen 1 und 32, und der Arbeitsspeicher zwischen 1 GB und 511 GB. Die Bandbreite zwischen den Servern und dem zentralen Switch wird mit Werten zwischen 0,1 GB/s und 1 GB/s modelliert. Da der Bitbrain-Datensatz keine Aussagen über den Festplattenspeicher der virtuellen Server enthält, werden hier gleichmäßige Werte zwischen 50 GB und 300 GB verwendet. Die genauen Wahrscheinlichkeitsverteilungen für die Prozessorkerne, den Arbeitsspeicher und die Bandbreite sind in [91] zu finden. Zur Evaluation der anwendungsfallspezifischen Anforderungen werden für jedes virtuelle Netzwerk zwischen 0 und 2 Standby-Server mit den jeweiligen primären Servern erzeugt. Zusätzlich fordert ein virtuelles Netzwerk mit einer Wahrscheinlichkeit von 20 % eine kurze Verzögerungszeit und jeder virtuelle Server ist mit einer Wahrscheinlichkeit von 20 % ein Hochleistungsserver.

Die dargestellten Datenpunkte einer Evaluation setzen sich aus dem Median von drei Messreihen zusammen, welche mit derselben Folge von Pseudozufallszahlen durchgeführt werden. So ist beispielsweise das fünfte (zufällig) generierte virtuelle Netzwerk aus Messreihe 1 für das Szenario A identisch mit dem fünften (zufällig) generierten virtuellen Netzwerk aus Messreihe 1 für das Szenario B. Genauso verhält es sich mit den pseudozufällig verteilten und festgelegten Ressourcen bzw. anwendungsspezifischen Werten. Alle Experimente werden auf einem Computer mit einem AMD Ryzen Threadripper 2990WX mit 32 Prozessorkernen und 128 GB Arbeitsspeicher ausgeführt. Als Betriebssystem wird Ubuntu 19.04, als Java Laufzeitumgebung OpenJDK 12 und als ILP-Löser Gurobi verwendet.

6.2 Statisches Anwendungsszenario

Im ersten Teil der Evaluation bildet ein statisches Anwendungsszenario die Grundlage für die einzelnen VNE-Probleme. Zur Lösung dieser Problemstellungen wird MdVNE, ein batchbasierter Ansatz, eingesetzt. Die MT- und ILP-Konfiguration wurde anhand der modellbasierten Problemdefinition und der zuvor vorgestellten per-Konstruktion-korrekten Methodik erstellt, wodurch MdVNE korrekte und optimale Lösungen findet. Als Vergleich wird ein manuell handoptimierter ILP-basierter Ansatz verwendet, da ILP-basierte Ansätze in der Literatur üblicher-

weise zur Beschreibung des VNE-Problems und als Referenz eingesetzt werden. Dabei wird sowohl für MdVNE als auch für den ILP-basierten Ansatz, im Weiteren als Basis-Ansatz bezeichnet, derselbe ILP-Löser, in derselben Konfiguration auf derselben Hardware verwendet. Auch die generierten ILP-Formulierungen für MdVNE und dem Basis-Ansatz sind vergleichbar aufgebaut und in einem ähnlichen Stil erstellt.

Folgende Freiheitsgrade sind in dieser Evaluation vorgesehen:

- die Größe des Substratnetzwerks (kSN und gSN),
- die Anzahl an virtuellen Netzwerken (1 oder 5) die gleichzeitig eingebettet werden (Batchgröße) und
- die Menge an Einschränkungen von zulässigen Einbettungen die berücksichtigt werden sollen (alle Anforderungen oder die Basis-Anforderungen).

Bei der Beachtung von allen Anforderungen (kurz AA) wird sichergestellt, dass sowohl die knoten- als auch die verbindungs- und anwendungsspezifischen Anforderungen aus Abschnitt 2.2 eingehalten werden. Bei den Basis-Anforderungen (kurz BA) hingegen werden nur die knoten- und verbindungsspezifischen Anforderungen (ohne die anwendungsspezifischen Anforderungen) beachtet. Somit werden die Ausfallsicherheit, die kurze Verzögerungszeit oder die Hochleistungsserver bei den Basis-Anforderungen nicht berücksichtigt. Zusätzlich soll in dieser Evaluation der aus der modellbasierten Problemdefinition abgeleitete und in eine Simulationsumgebung eingebettete Quellcode bezüglich der (theoretisch) bewiesenen Korrektheit und Optimalität für (i)MdVNE untersucht werden. Dazu werden wir diskutieren, ob die durchgeführten Maßnahmen z.B. zur Qualitätssicherung des erzeugten Quellcodes die Wahrscheinlichkeit für inkorrekte oder suboptimale Ergebnisse verringert und ob sich dies durch die evaluierten Daten bestätigen lässt. Zur Untersuchung der beiden Evaluationsziele werden die folgenden Forschungsfragen (FF), kategorisiert nach den Evaluationszielen, untersucht:

Evaluationsziel A (Effizienz):

FF A.1 (Skalierbarkeit): Welchen Einfluss hat die *Modellgröße* auf die Laufzeit zur Lösung des VNE-Problems?

FF A.2 (Anforderungen): Hat die *Anzahl an Anforderungen* einen Einfluss auf die Laufzeit zur Lösung des VNE-Problems?

FF A.3 (Batchgröße): Wie beeinflusst die *Batchgröße* die Laufzeit zur Lösung des VNE-Problems?

Evaluationsziel B (Effektivität):

FF B.1 (Korrektheit): In welchem Umfang wird die *Korrektheit* bei MdVNE sichergestellt?

FF B.2 (Optimalität): In welchem Umfang wird die *Optimalität* bei MdVNE sichergestellt?

6.2.1 *Versuchsaufbau*

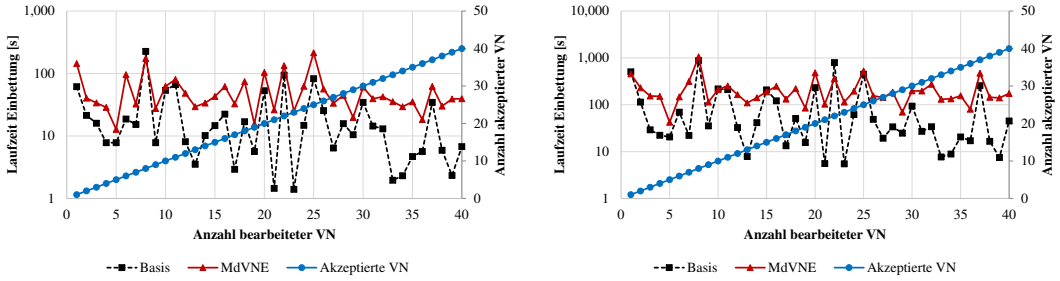
Da es sich bei dieser Evaluation um einen statischen Anwendungsfall handelt, werden in dem vorher beschriebenen Versuchsaufbau (siehe Abschnitt 6.1) 40 zufällig generierte virtuelle Netzwerke eingebettet. Dabei werden entweder alle virtuellen Netzwerke nacheinander (Batchgröße eins oder kurz BG1) oder fünf virtuelle Netzwerke gleichzeitig (Batchgröße fünf oder kurz BG5) im Substratnetzwerk platziert. Bei der Einbettung müssen zusätzlich entweder alle Anforderungen (kurz AA) oder nur die Basis-Anforderungen (kurz BA) eingehalten werden.

6.2.2 *FF A.1 - Skalierbarkeit*

> Welchen Einfluss hat die *Modellgröße* auf die Laufzeit zur Lösung des VNE-Problems?

Bei dieser Forschungsfrage wird die Skalierbarkeit von MdVNE im Vergleich zum ILP-basierten Basis-Ansatz zur Lösung eines VNE-Problems in einem kleinen oder großen Rechenzentrum untersucht und diskutiert. Dazu werden alle virtuellen Netzwerke unter Berücksichtigung aller Anforderungen (AA) nacheinander eingebettet (BG1). In den Diagrammen in Abb. 6.1 sind die Ergebnisse zur Untersuchung der Laufzeit zur Lösung des VNE-Problems für ein kleines (Abb. 6.1a) und für ein großes Substratnetzwerk (Abb. 6.1b) über die Anzahl an bearbeiteten virtuellen Netzwerken (VN) abgebildet. Auf der primären logarithmisch skalierten x-Achse ist die Gesamtlaufzeit zur Einbettung eines virtuellen Netzwerkes mit allen Anforderungen in Sekunden angegeben. Die y-Achse zeigt dazu die Anzahl an zu bearbeitenden virtuellen Netzwerken (VN) an. Neben dem MdVNE-Ansatz (MdVNE), welcher als durchgezogene rote Linie mit einem Dreieck für die Datenpunkte dargestellt ist, wird der Basis-Ansatz (Basis) als gepunktete schwarze Linie mit Quadraten als Datenpunkte visualisiert. Zusätzlich wird auf der sekundären Achse die Anzahl akzeptierter, d.h. erfolgreich eingebetteter, virtueller Netzwerke dargestellt. In Abb. 6.3 werden die Laufzeiten von MdVNE und dem Basis-Ansatz für ein kleines und großes Substratnetzwerk in einem Diagramm gegenübergestellt. Dabei repräsentieren die gepunkteten Linien den Basis- und die durchgezogenen Linien den MdVNE-Ansatz. In Abb. 6.2 werden die Summen aller Datenpunkte für die Laufzeit der Einbettungen aller 40 virtuellen Netzwerke aus Abb. 6.1 dargestellt. Neben der reinen Laufzeit zur Lösung der ILP-Formulierung werden auch der MT-Anteil bei MdVNE bzw. die sonstigen Anteile für den Basis-Ansatz visualisiert. Die Gesamtlaufzeit zur Einbettung der virtuellen Netzwerke ergibt sich aus der Summe dieser beiden Anteile.

Anhand der Abb. 6.1a und 6.1b wird ersichtlich, dass sowohl MdVNE als auch der Basis-Ansatz alle virtuellen Netzwerke nacheinander einbetten konnten. Bei genauerer Betrachtung der Kurven für die Laufzeit zur Einbettung der virtuellen Netzwerke erkennt man, dass sich die Laufzeit bei beiden Ansätze in ähnlicher Art und Weise verhält. So wird beispielsweise zum Lösen des VNE-Problems für das achte virtuelle Netzwerk in einem kleinen und großen Substratnetzwerk sowohl bei MdVNE als auch beim Basis-Ansatz mehr Zeit benötigt als beispielsweise



(a) Kleines Substratnetzwerk **(b)** Großes Substratnetzwerk

Abbildung 6.1: Laufzeit zum Lösen des VNE-Problems mit allen Anforderungen und einer Batchgröße von eins

für das fünfte virtuelle Netzwerk. Man könnte sagen, die Kurven verhalten sich ähnlich in Bezug auf die Problemstellung, wobei MdVNE einen zusätzlichen Zeitversatz durch den MT-Anteil aufweist. Die stark schwankenden Laufzeiten (z.B. zwischen der Einbettung des fünften und achten virtuellen Netzwerks) lassen sich durch die unterschiedlichen konkreten Rahmenbedingungen erklären, die zu diesen Zeitpunkten vorliegen. Diese Rahmenbedingungen beinhalten schon vorhandene Einbettungen und deren Verteilung im Substratnetzwerk. Somit kann in einigen Fällen sehr schnell eine Lösung gefunden werden (z.B. wenn alle virtuellen Server auf einem Substratserver oder in einem Rack platziert werden können) oder die Berechnung der optimalen Lösung benötigt mehr Zeit (z.B. wenn die virtuellen Server über mehrere Racks verteilt werden müssen). Da die Schwankungen bei MdVNE und dem Basis-Ansatz ähnlich verlaufen, ist dies ein Indiz dafür, dass die ILP-Formulierungen und die gefundenen Einbettungen vergleichbar zueinander sind und vergleichbar durch den ILP-Löser gelöst werden können. Anhand von Abb. 6.2 kann man erkennen, dass MdVNE für das kleine Substratnetzwerk 218 % und für das große Substratnetzwerk 183 % mehr Zeit im Vergleich zum Basis-Ansatz benötigt. Dabei ist die Zeit zum Lösen der ILP-Formulierung (ILP-Anteil) mit 108 % für das kleine und 82 % für das große Netzwerk vergleichbar mit dem Basis-Ansatz. Somit zeigt sich, dass der zusätzliche MT-Schritt für dieses statische VNE-Problem keinen Laufzeitvorteil mit sich bringt, da die zusätzliche Laufzeit durch den MT-Anteil den ILP-Anteil nicht in gleichem Maße verringert.

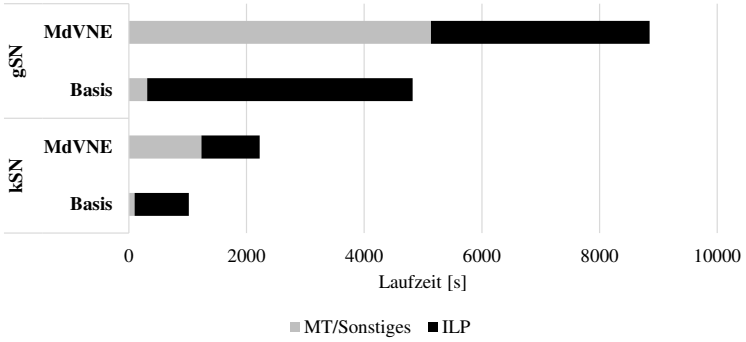


Abbildung 6.2: Summe über die Laufzeit für alle Einbettungen aus Abb. 6.1

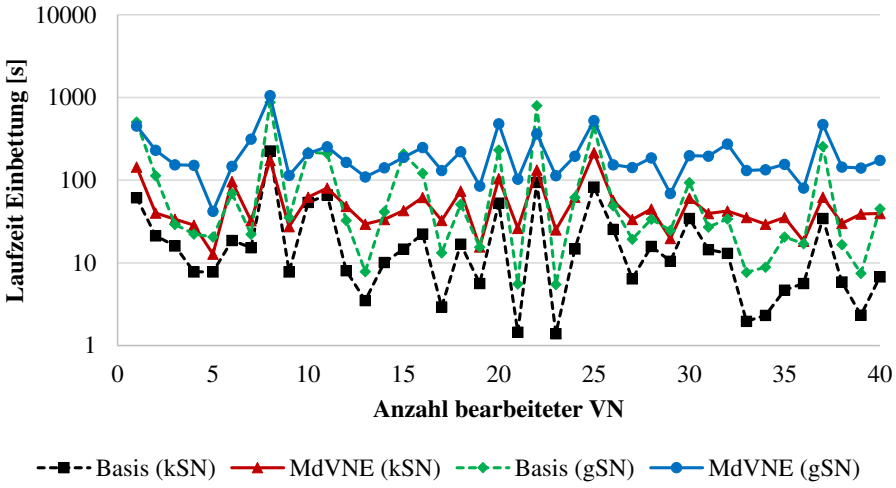


Abbildung 6.3: Vergleich für ein kleines und großes Substratnetzwerk mit allen Anforderungen und einer Batchgröße von eins

Wenn man diese Kurven zusammen in Abb. 6.3 vergleicht, kann man erkennen, dass eine Erhöhung der Modellgröße mit einer Erhöhung der Laufzeit einhergeht, wobei die Form der Kurve weitgehend erhalten bleibt. Dabei erhöht sich die Laufzeit für den Basis-Ansatz ungefähr um den Faktor 4,7, wo hingegen sich bei MdVNE die Laufzeit (nur) um den Faktor 4 erhöht, was sich auch ungefähr im ILP-Anteil der beiden Ansätze wieder spiegelt. Dieses Verhalten kann durch eine Verringerung des MT-Anteils erklärt werden, da sich die MT-Laufzeit direkt proportional mit der Anzahl an virtuellen und Substratelementen erhöht. Da sich die Modellgröße in Bezug auf das Substratnetzwerk von 80 auf 120 Servern um 50 % erhöht, können wir erkennen, dass die Laufzeit zur Lösung dieser Problemstellung sehr wahrscheinlich erwartungsgemäß ein exponentielles Verhalten besitzt.

Die Forschungsfrage FF A.1 zur Untersuchung der Skalierbarkeit beantworten wir daher wie folgt: Die Modellgröße hat einen großen Einfluss auf die Laufzeit zur Lösung des VNE-Problems, da sich gezeigt hat, dass sich bei einer Erhöhung der Modellgröße um 50 % die Laufzeit für beide Ansätze ungefähr um den Faktor 4 erhöht. Dabei konnte MdVNE eine leicht verringerte Erhöhung der Laufzeit beim großen Substratnetzwerk mit einem Faktor von 4 gegenüber dem Basis-Ansatz mit einem Faktor von 4,7 erreichen. Allerdings benötigte MdVNE bei einem kleinen und großen Substratnetzwerk ungefähr doppelt so viel Zeit zum Lösen der Einbettung wie der Basis-Ansatz.

6.2.3 *FF A.2 - Anforderungen*

> Hat die *Anzahl an Anforderungen* einen Einfluss auf die Laufzeit zur Lösung des VNE-Problems?

Bei dieser Forschungsfrage wird untersucht, wie sich die Anzahl an (einschränkenden) Anforderungen auf die Laufzeit zur Lösung des VNE-Problems auswirkt. Dazu wird das VNE-Problem im ersten Fall mit allen Anforderungen und im zweiten Fall nur mit den Basis-Anforderungen gelöst. Die 40 virtuellen Netzwer-

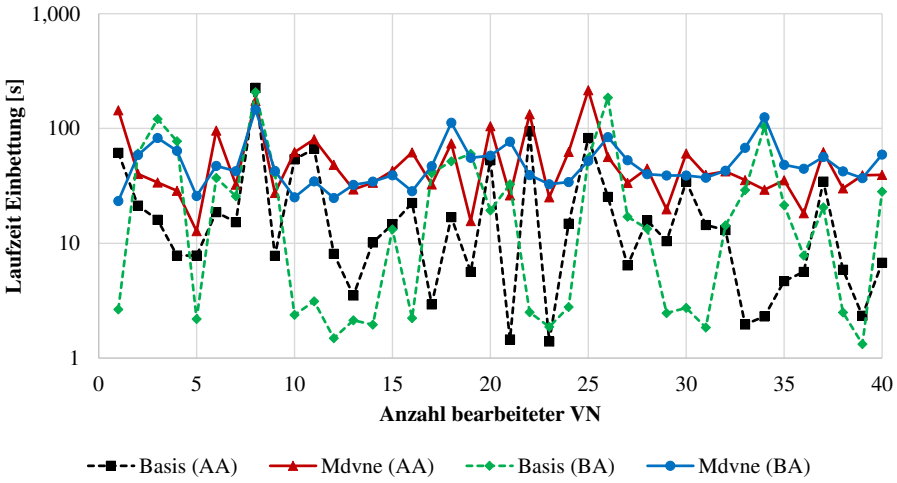


Abbildung 6.4: Laufzeit zum Lösen des VNE-Problems für ein kleines Substratnetzwerk mit einer Batchgröße von eins

ke werden dabei einzeln nacheinander in das kleine bzw. das große Substratnetzwerk eingebettet (BG1).

In Abb. 6.4 sind die Ergebnisse dieser Messreihen für das kleine Substratnetzwerk und in Abb. 6.5 für das große Substratnetzwerk dargestellt. Diese Diagramme zeigen die Laufzeit zur Einbettung eines virtuellen Netzwerks über der Anzahl von bearbeiteten virtuellen Netzwerken. Sowohl beim kleinen als auch beim großen Substratnetzwerk konnten alle virtuellen Netzwerke bei beiden Ansätzen erfolgreich eingebettet werden. Die gepunkteten Linien stellen wie zuvor auch den Basis-Ansatz und die durchgezogenen Linien den MdVNE-Ansatz jeweils für alle Anforderungen (AA) und die Basis-Anforderungen (BA) dar. In Abb. 6.6 sind für die beiden Diagramme aus Abb. 6.4 und 6.5 wieder die Summen für die Laufzeiten, unterteilt in die beiden Anteile MT und ILP, zur Einbettung von allen virtuellen Netzwerken für MdVNE und dem Basis-Ansatz dargestellt.

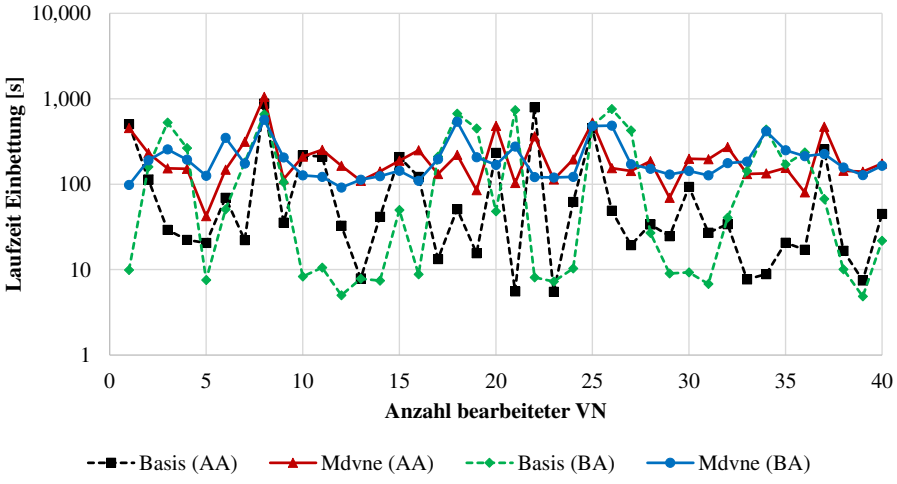


Abbildung 6.5: Laufzeit zum Lösen des VNE-Problems für ein großes Substratnetzwerk mit einer Batchgröße von eins

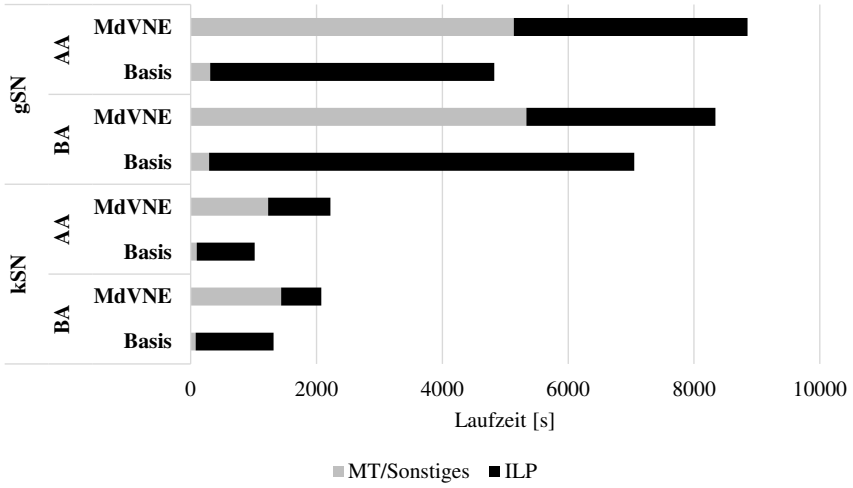


Abbildung 6.6: Summe über die Laufzeit für alle Einbettungen aus Abb. 6.4 und 6.5

Zunächst fällt in den dargestellten Grafiken auf, dass der grundlegende Verlauf der Kurven (mit Höhen und Tiefen) zwischen MdVNE und dem Basis-Ansatz sowohl bei allen Anforderungen als auch bei den Basis-Anforderungen sehr ähnlich ist. Dies stellt ein Indiz dafür dar, dass beide Ansätze ähnlich komplex zu lösende ILP-Formulierungen erstellen und vergleichbare Lösungen gefunden haben, was durch stichprobenartige Überprüfung bestätigt wurde. In Abb. 6.6 erkennt man, dass sich bei MdVNE sowohl die Gesamtlaufzeit als auch die Zeit zum Lösen der ILP-Formulierung zwischen BA und AA von 2076 s auf 2223 s erhöht (ca. 7 %), wo hingegen sich beim Basis-Ansatz die Laufzeit von 1320 s auf 1019 s (ca. 23 %) für das kleine Substratnetzwerk verringert. Diese Tendenz kann auch für den ILP-Anteil oder das große Substratnetzwerk bei MdVNE oder dem Basis-Ansatz nachgewiesen werden. Bei genauer Betrachtung des MT- und ILP-Anteils bei Erhöhung der Anforderungen für MdVNE stellt man fest, dass sich der MT-Anteil bei Erhöhung der Anzahl an Anforderungen geringfügig verringert (Differenz zwischen ILP-Anteil und Gesamtlaufzeit), wo hingegen sich der ILP-Anteil erhöht. Dies lässt sich durch die folgenden Punkte erklären:

- Bei Erhöhung der Anzahl an Anforderungen werden mehr und teilweise komplexere MT-Regeln verwendet, wodurch sich ungültige Platzierungskandidaten bzw. (in)korrekte Übereinstimmungen durch die zusätzlichen Einschränkungen effizienter (schneller) finden lassen.
- Durch die zusätzlichen Anforderungen konnten nicht signifikant mehr Platzierungskandidaten verworfen werden, wodurch sich die Menge an ILP-Variablen nur geringfügig verändert.
- Durch die erhöhte Anzahl an komplexeren MT-Regeln werden durch den MT-ILP-Konverter (siehe Abb. 5.1) mehr ILP-Ungleichungen generiert, wodurch das Finden einer optimalen Lösung für den ILP-Löser erschwert wird.

Beim Basis-Ansatz hingegen kann ein gegenteiliges Verhalten beobachtet werden, da sich durch das Hinzufügen von zusätzlichen Anforderungen die Laufzeit zum

Lösen des VNE-Problems verringert. Dies lässt sich dadurch erklären, dass die zusätzlichen Anforderungen die Menge an möglichen Lösungen (stark) einschränkt, wodurch der ILP-Löser in der Lage ist, schneller eine optimale Lösung zu finden. Dieses gegensätzliche Verhalten des ILP-Lösers sollte zukünftig genauer untersucht werden, da nicht nur die Menge von ILP-Ungleichungen, sondern auch der Aufbau in Kombination schon vorhandener Ungleichungen das Lösen einer Problemstellung erschweren oder vereinfachen kann. Beim Vergleich der ILP-Anteile zwischen MdVNE und dem Basis-Ansatz fällt auf, dass MdVNE die Zeit zum Lösen des VNE-Problems um bis 56 % (für gSN) verringern konnte.

Die Forschungsfrage FF A.2 zur Untersuchung der Laufzeit in Bezug auf die Anzahl an Anforderungen beantworten wir daher wie folgt: Im Fall von MdVNE hat die Anzahl an Anforderungen nur einen geringen Einfluss auf die Gesamtlaufzeit zur Lösung des VNE-Problems. Zusätzliche Anforderungen haben die Laufzeit hier nur um ca. 7 % erhöht. Beim Basis-Ansatz hingegen konnte mit ca. 23 % eine signifikante Verringerung der Laufzeit durch die Integration von zusätzlichen Anforderungen festgestellt werden. Weiter konnten wir beobachten, dass durch MdVNE sich der ILP-Anteil im Vergleich zum Basis-Ansatz um bis zu 56 % verringert hat, wobei die Gesamtlaufzeit in diesem Fall trotz der Einsparung durch den ILP-Anteil um ca. 18 % höher ist als beim Basis-Ansatz. Dies ist auf den Mehraufwand bei der MT-Regelausführung und den zugrundeliegenden Technologien zurückzuführen, welche noch Optimierungspotenziale für Effizienzgewinne aufweisen.

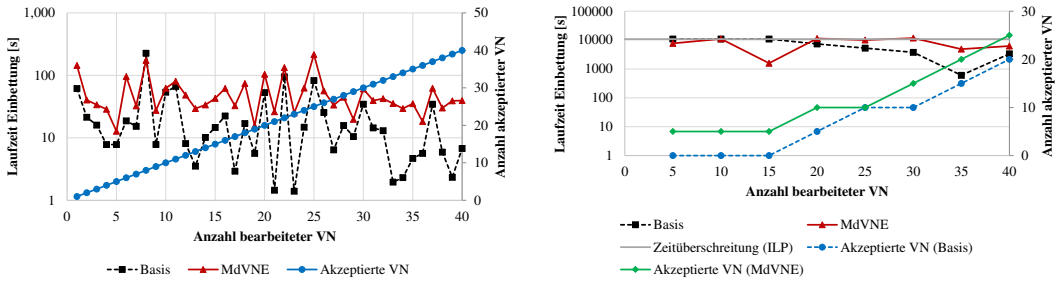
6.2.4 *FF A.3 - Batchgröße*

> Wie beeinflusst die *Batchgröße* die Laufzeit zur Lösung des VNE-Problems?

Diese Forschungsfrage untersucht den Einfluss der Batchgröße auf die Laufzeit zur Lösung des VNE-Problems. Dazu wird ein kleines Substratnetzwerk mit allen Anforderungen simuliert und jeweils ein oder fünf virtuelle Netzwerke gleichzeitig eingebettet (Batchgröße eins bzw. fünf). Bei der Einbettung von fünf virtuellen Netzwerken gleichzeitig, muss eine Lösung für alle fünf virtuellen Netzwerke gefunden werden. Es ist hierbei nicht zulässig, dass nur eine Teilmenge der fünf virtuellen Netzwerke eingebettet wird.

Die Ergebnisse der Messungen sind in Abb. 6.7 dargestellt. Die Diagramme zeigen jeweils die Laufzeit zur Lösung des VNE-Problems auf der primären und die Anzahl der akzeptierenden bzw. erfolgreich eingebetteten virtuellen Netzwerke auf der sekundären Achse jeweils über der Anzahl von bearbeiteten virtuellen Netzwerken. Neben den Datenpunkten für die Laufzeit des Basis- und MdVNE-Ansatzes findet sich die Anzahl der akzeptierten virtuellen Netzwerke und in Abb. 6.7b zusätzlich auch die maximale Laufzeit für den ILP-Löser (Zeitüberschreitung ILP). Wenn der Wert für die Zeitüberschreitung (Ztü.) von 3 Stunden durch den ILP-Löser erreicht wird, bricht der ILP-Löser alle Berechnungen ab und gibt das bisher beste Ergebnis zurück, falls schon ein Ergebnis existiert. In Abb. 6.8 sind die Summen über alle Einbettungen zusammengefasst darge-

stellt, wobei die Einbettungen, die in die Zeitüberschreitung gelaufen sind, nicht berücksichtigt werden.



(a) Batchgröße eins **(b)** Batchgröße fünf

Abbildung 6.7: Laufzeit zum Lösen des VNE-Problems für ein kleines Substratnetzwerk mit allen Anforderungen

Man erkennt, dass bei einer Batchgröße von eins jedes virtuelle Netzwerk sowohl bei MdVNE als auch beim Basis-Ansatz eingebettet werden kann. Im Fall der Batchgröße von fünf ist dies nicht mehr möglich. So kann beispielsweise der Basis-Ansatz die ersten 15 virtuellen Netzwerke und auch die Netzwerke 25 bis 30 nicht einbetten. Bei den ersten 15 virtuellen Netzwerken liegt dies an der Zeitüberschreitung, da der ILP-Löser bis dahin keine Lösung finden konnte. Die Netzwerke 25 bis 30 können durch die Wahl von vorangegangenen Einbettungen nicht platziert werden. Auch MdVNE läuft insgesamt dreimal in die Zeitüberschreitung (bei den Netzwerken 10, 20 und 30), allerdings ist bei Netzwerk 30 der ILP-Löser trotz der Zeitüberschreitung in der Lage eine Lösung zu finden. Bei einer stichprobenartigen Untersuchung konnte festgestellt werden, dass die Laufzeit zum Finden einer Lösung durch den ILP-Löser teilweise mehr als zwei Tage betrug.

Beim Vergleich der Gesamtlaufzeiten fällt auf, dass MdVNE ca. 14-mal mehr Zeit benötigt, um fünf virtuelle Netzwerke gleichzeitig einzubetten, und der Basis-Ansatz ca. 20-mal mehr Zeit. Dies legt nahe, dass die Laufzeit exponentiell mit der Batchgröße wächst. Dabei weist MdVNE im Vergleich zum Basis-Ansatz prozentual eine geringere Erhöhung der Laufzeit auf und benötigt somit statt 118 % (für BG1) nur ca. 50 % mehr Zeit (für BG5) als der Basis-Ansatz. Zusätzlich verringert sich der prozentuale MT-Anteil von MdVNE bei Erhöhung der Batchgröße. So betrug der MT-Anteil bei Batchgröße eins ca. 55 % und verringerte sich bei

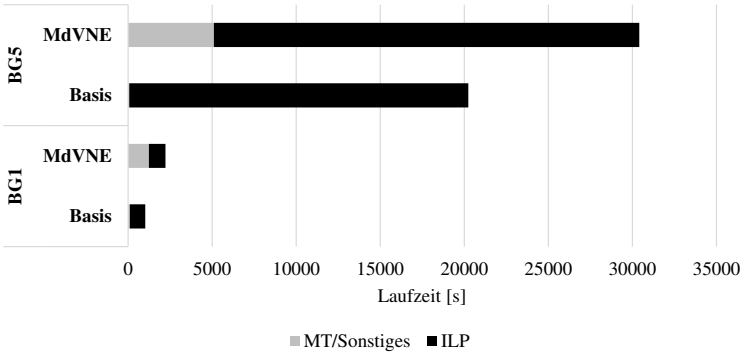


Abbildung 6.8: Summe über die Laufzeit aller Einbettungen aus Abb. 6.7

Batchgröße fünf auf ca. 17 %. Diese Beobachtungen können dabei folgendermaßen erklärt werden:

- Der Aufbau des Substratnetzwerks muss für die Batchgröße eins fünfmal durchgeführt werden, um fünf virtuelle Netzwerke einzubetten. Somit kann durch die Batchgröße fünf der Aufbau des Substratnetzwerkes viermal eingespart werden.
- Die Laufzeit zur Lösung der ILP-Formulierung wächst exponentiell mit der Anzahl an Variablen, die proportional zu der Anzahl an virtuellen Elementen multipliziert mit den Substratelementen wächst ($|VN| * |SN|$).
- Die MT-Laufzeiten wachsen polynomiell mit $|VN| * |SN|$.

Die Forschungsfrage FF A.3 zur Untersuchung der Laufzeit in Bezug auf die Batchgröße beantworten wir daher wie folgt: Die Batchgröße hat einen erheblichen Einfluss auf die Laufzeit, da sich die Laufzeit, ausgehend von den gezeigten Daten, sehr wahrscheinlich exponentiell erhöht. So benötigt der Basis-Ansatz ca. 20-mal mehr Zeit für eine Batchgröße von fünf im Vergleich zu einer Batchgröße von eins, wo hingegen MdVNE nur ca. 14-mal mehr Zeit benötigt. Zusätzlich konnte der MT-Anteil bei Erhöhung der Batchgröße signifikant von ca. 55 % auf ca. 17 % verringert werden.

6.2.5 *FF B.1 - Korrektheit*

In welchem Umfang wird die *Korrektheit* bei MdVNE sichergestellt?

Bei dieser Forschungsfrage wird diskutiert, welche Methoden und Techniken verwendet wurden, um die Korrektheit der Ergebnisse zu gewährleisten. Insgesamt wurden zur Sicherung der Korrektheit die folgenden Methoden angewendet:

(a) Nutzung der Konstruktionsmethodik aus Abschnitt 4.3.

(b) Vergleich der Ergebnisse von MdVNE mit dem Basis-Ansatz.

(c) Konsistenzprüfung während der Einbettung der virtuellen Netzwerke in das Substratnetzwerk.

(d) Einsatz von Unit-Tests.

Durch den Einsatz der in dieser Arbeit vorgestellten per-Konstruktion-korrekten Methodik (a) wird sichergestellt, dass die abgeleiteten MT- und ILP-Konfigurationen für MdVNE korrekt sind. Somit wird schon während des Entwicklungsprozesses die Korrektheit der Ergebnisse mit beachtet.

Weiter wurde durch den Vergleich der Ergebnisse zwischen MdVNE und dem Basis-Ansatz (b) gezeigt, dass alle virtuellen Netzwerke analog zueinander eingebettet wurden und die Werte der Zielfunktion identisch sind. Zusätzlich konnte durch die redundante Implementierung desselben Problems mithilfe einer anderen Technologie die Wahrscheinlichkeit für Fehler in MdVNE verringert werden.

Durch die Konsistenzprüfungen während und nach den Einbettungen (c) wurden zusätzlich alle Anforderungen nochmals verifiziert, wodurch beispielsweise

sichergestellt wird, dass alle virtuellen Elemente eingebettet und keine Substratressourcen überbucht werden. Diese Konsistenzprüfungen bieten zusätzliche Sicherheit, dass die gefundenen Lösungen in Bezug auf die Anforderungen korrekt sind.

Zusätzlich wurden durch den Einsatz von Unit-Tests (d) weitere Fehlerquellen ausgeschlossen, da manuell im Vorfeld berechnete Lösungen für unterschiedliche Netzwerktopologien (1-Tier, 2-Tier und Fat-Tree) und den daraus resultierenden Problemstellungen als Vergleich dienen. Diese Tests wurden hierbei sowohl für MdVNE als auch für den Basis-Ansatz verwendet. In Summe wurden hierfür 330 Unit-Tests erstellt, um die Korrektheit (und Optimalität) zu gewährleisten.

Die Forschungsfrage FF B.1 zur Sicherstellung der Korrektheit beantworten wir daher wie folgt: Durch den Einsatz der in Abschnitt 4.3 per-Konstruktion-korrekten Methodik in Kombination mit einer redundanten Implementierung als Vergleich, der Überprüfung der Anforderung nach Umsetzung der gefundenen Lösungen und dem Einsatz von Unit-Tests, kann die Wahrscheinlichkeit für inkorrekte Ergebnisse stark verringert werden.

6.2.6 *FF B.2 - Optimalität*

In welchem Umfang wird die *Optimalität* bei MdVNE sichergestellt?

Damit MdVNE optimale Lösungen findet und zurückgibt, wurde zuerst die Zielfunktion mit den dafür notwendigen ILP-Variablen und -Ungleichungen unter Beachtung der per-Konstruktion-korrekten Methodik erstellt. Dadurch wird die Optimalität schon während des gesamten Entwicklungsprozesses sichergestellt. Zusätzlich wurden die Werte der Zielfunktion von MdVNE mit denen des Basis-Ansatzes verglichen, wodurch die Wahrscheinlichkeit minimiert werden konnte, dass MdVNE suboptimale Lösungen hinsichtlich des Optimierungsziels zurückgibt. Hierbei konnten wir feststellen, dass MdVNE und der Basis-Ansatz dieselben Werte für die Zielfunktion als optimal zurückgegeben haben. Dies bedeutet nicht, dass MdVNE und der Basis-Ansatz dieselben Einbettungen als Lösungen gefunden haben, da es (normalerweise) mehrere optimale Lösungen für eine Menge von virtuellen Netzwerken gibt. Ein Unterschied in den Werten für die Zielfunktion ließ sich erst beobachten als die im Substratnetzwerk verfügbaren Ressourcen gegen Null gingen, da nun vergangene Einbettungen einen höheren Einfluss auf aktuelle mögliche Einbettungen besitzen und somit andere optimale Lösungen gefunden werden.

Die Forschungsfrage FF B.2 zur Sicherstellung der Optimalität beantworten wir daher wie folgt: Die Optimalität wird durch die per-Konstruktion-korrekte Methodik und dem Vergleich der Werte der Zielfunktion für die Lösungen von MdVNE mit dem Basis-Ansatz sichergestellt.

6.2.7 *Gefährdung der Validität der Ergebnisse*

Für MdVNE und deren MT- bzw. ILP-Konfiguration ist es schwierig zu überprüfen und sicherzustellen, ob bzw. dass die gefundenen Lösungen korrekt und

optimal sind, da getroffene Einbettungsentscheidungen Auswirkungen auf zukünftige Einbettungen haben. Dadurch ist ein Vergleich bezüglich der tatsächlich gefundenen Lösungen zwischen MdVNE und dem Basis-Ansatz nur begrenzt aussagekräftig. Auch ist die Auswahl eines optimalen Kandidaten aus der Menge von allen optimalen Lösungen nicht weiter spezifiziert, sodass für verschiedene oder sogar gleiche Messreihen unterschiedliche Einbettungen gefunden werden können. Zur Reduktion des Risikos für inkorrekte Lösungen, verwenden wir Konsistenzprüfungen während der Laufzeit, eine redundante ILP-basierte Implementierung und Unit-Tests.

Zur Erstellung des Anwendungsszenarios für diese Evaluation wurden vergleichbare Evaluationen aus anderen Veröffentlichungen und reale Wahrscheinlichkeitsverteilungen von Anwendungen aus Rechenzentren verwendet. Bei Betrachtung der Evaluationen aus den Veröffentlichungen aus Abschnitt 2.2.3 und weiteren Veröffentlichungen aus diesem Bereich (siehe auch [56]) konnten wir feststellen, dass vorrangig Tier-basierte Netzwerkstrukturen als Substratnetzwerke verwendet wurden. Am häufigsten wurden dabei (Google) Fat-Tree (3-Tier) Topologien mit 16 oder 128 Substratservern eingesetzt. Bei ersten Untersuchungen hat sich allerdings gezeigt, dass durch die enorme Menge an möglichen Pfaden in einem 3-Tier-Netzwerk das VNE-Problem für einen ILP-Löser nicht in einer annehmbaren Zeit gelöst werden kann. Daher wird in dieser Evaluation eine 2-Tier-Netzwerktopologie mit 80 und 120 Substratservern verwendet, wodurch die grundlegenden Eigenschaften durch die Reduzierung einer Netzwerkebene im Vergleich zu einem 3-Tier-Netzwerk erhalten bleiben, aber wesentlich weniger Substratpfade betrachtet werden müssen. Für die virtuellen Netzwerktopologien wurde in der Mehrheit der Veröffentlichungen, falls spezifische Topologien benannt wurden, eine sternförmige Netzwerktopologie verwendet. Aus diesem Grund haben wir uns ebenfalls für sternförmige Netzwerktopologien der virtuellen Netzwerke entschieden. Damit die Anforderungen an die virtuellen Elemente so realitätsnah wie möglich sind, orientierten wir uns an dem statistischen Datensatz für Anwendungen in einem Rechenzentrum, welcher Wahrscheinlichkeitsfunktionen von real gemessenen Werten für die angeforderten und tatsächlich verwendeten Ressourcen wie z.B. CPU-Kerne, Arbeitsspeicher und Bandbreite zur Verfügung stellt [91]. Damit die Auswirkung der zufällig gewählten Werte reduziert wird, wurden alle Experimente dreimal wiederholt und daraus der Median berechnet. Die Anzahl der Wiederholungen wurde wegen der langen Laufzeit einzelner Experimente begrenzt.

Neben der Nutzung von ILP-Technologien zur Beschreibung und Lösung des VNE-Problems könnten auch anderen Techniken beispielsweise aus dem Bereich der Bedingungserfüllungsprobleme oder von Erfüllbarkeits-Modulo-Theorien zum Einsatz kommen. Dabei könnten SAT- oder SMT-Löser zur Lösung der Problemstellung genutzt werden. In dieser Problemdomäne stellt ILP eine etablierte Technologie zur Beschreibung dieser Problemstellung dar. So wurde die hier verwendete ILP-Formulierung von [85] inspiriert und kontinuierlich in Zusammenarbeit mit Experten weiter verbessert. Da die Laufzeit zum Lösen der ILP-Formulierung stark von der Wahl des ILP-Lösers und der jeweiligen Version abhängt, haben wir uns für Gurobi, einen etablierten, in der Industrie und in verschiedenen Ver-

öffentlichungen (z.B. [105, 78, 63, 118]) eingesetzten und dem Stand der Technik entsprechenden ILP-Löser entschieden [67]. Bei Untersuchungen mit Cplex [15] für dynamische Anwendungsszenarien konnte festgestellt werden, dass Gurobi eine (leicht) schnellere Laufzeit zur Lösung von sich inkrementell ändernden Problemstellungen aufwies [8]. Damit für die gesamte Evaluation, auch für das dynamische Anwendungsszenario, derselbe ILP-Löser zum Einsatz kommt, fiel die Wahl daher auf Gurobi.

6.2.8 *Zusammenfassung*

In dieser Evaluation für ein statisches Anwendungsszenario konnten wir zeigen, dass der in dieser Arbeit vorgestellte MdVNE-Ansatz ähnlich zu einem ILP-basierten Ansatz skaliert, wobei durch den MT-Anteil die Laufzeit zur Lösung der Problemstellung ungefähr (konstant) verdoppelt wurde. Allerdings konnte gezeigt werden, dass durch MdVNE der ILP-Anteil gegenüber dem Basis-Ansatz verringert wird, was sich besonders bei der veränderten Anzahl an Anforderungen gezeigt hat. Bei einer Batchgröße von fünf konnte bei MdVNE der MT-Anteil signifikant von ca. 55 % auf 17 % verringert werden, wodurch der ILP-Anteil die dominierte Größe bei der Lösung des VNE-Problems darstellt und annähernd vergleichbar ist zum Basis-Ansatz. Insgesamt wurde deutlich, dass ein handgeschriebenes und -optimiertes (Low-Level) Programm für die Generierung der ILP-Formulierung noch deutlich effizienter ist als ein modellbasierter Ansatz, wobei hierbei noch Optimierungspotentiale bei den im Mustererkenner eingesetzten Technologien bestehen. Allerdings konnte durch die Verwendung der per-Konstruktion-korrekten Methodik, der deklarativen MT-Regeln und der weitgehenden Automatisierung bei der Quellcodegenerierung die Entwicklungszeit verkürzt und die Qualität in Bezug auf den Quellcode, die Wartbarkeit und die Anpassbarkeit erhöht werden.

6.3 Dynamisches Anwendungsszenario

Im zweiten Teil der Evaluation wird ein dynamisches Anwendungsszenario simuliert, um iMdVNE und die damit eingeführten inkrementellen Erweiterungen für MdVNE näher zu untersuchen. In diesem Anwendungsszenario werden wir uns auf das Hinzufügen von virtuellen Netzwerken und das Löschen von Substratservern konzentrieren, wobei sowohl die notwendige (NM) als auch die optimierende Migration (OM) zum Einsatz kommen. Bei der notwendigen Migration werden nur die verletzten Anforderungen repariert, was im Falle des Löschens eines Substratservers in der Migration der darauf eingebetteten virtuellen Server resultiert. Alle anderen Einbettungen werden nicht weiter beachtet, wodurch auch keine Änderungen von bestehenden Einbettungen möglich sind. Damit wird durch die notwendige Migration auch keine optimale Lösung garantiert. Bei der optimierenden Migration hingegen wird nach jedem Ereignis sichergestellt, dass eine optimale Lösung gefunden und umgesetzt wurde. Dazu wird das komplette VNE-Problem erneut gelöst und, falls erforderlich, Migrationen mit integriert. Falls virtuelle Netzwerke dadurch nicht mehr eingebettet werden können, werden

diese Netzwerke abgelehnt. Zur Berechnung der Migrationskosten für jeden virtuellen Server werden die Ressourcen dieses virtuellen Servers aufsummiert und mit in die Kostenfunktion integriert, falls dieser Server migriert wird. Als Vergleichsalgorithmus wird ein manuell erstelltes Java Programm verwendet, welches inkrementell die ILP-Formulierung generiert und aktualisiert.

Beispiel 6.1 (Berechnung der Migrationskosten)**.** Die Migrationskosten für den virtuellen Server `vsr1` aus Abb. 5.3a ist zwei, da dieser Server zwei Prozessorkerne besitzt.

Zur Untersuchung der beiden Evaluationsziele werden wir die folgenden Forschungsfragen untersuchen. Dabei wird Evaluationsziel A jeweils für die notwendige und die optimierende Migration getrennt voneinander betrachtet und das Evaluationsziel B für iMdVNE im Gesamten.

Evaluationsziel A (Effizienz):

FF A.1 (Skalierbarkeit): Welchen Einfluss hat die *Modellgröße* auf die Laufzeit zur Lösung des VNE-Problems?

FF A.2 (Anforderungen): Hat die *Anzahl an Anforderungen* einen Einfluss auf die Laufzeit zur Lösung des VNE-Problems?

Evaluationsziel B (Effektivität):

FF B.1 (Korrektheit und Optimalität): In welchem Umfang wird die *Korrektheit und Optimalität* bei iMdVNE sichergestellt?

6.3.1 *Versuchsaufbau*

In dieser Evaluation wird ein dynamisches Anwendungsszenario simuliert, welches auf dem Versuchsaufbau aus Abschnitt 6.1 basiert. Weiter soll diese Evaluation, soweit möglich, vergleichbar sein zur Evaluation eines statischen Anwendungsszenarios, um hier die Unterschiede aufzuzeigen, die sich durch Integration der inkrementellen Technologien ergeben. Somit werden in dieser Evaluation erneut 40 virtuelle Netzwerke nacheinander (Batchgröße eins) eingebettet. Zur weiteren Untersuchung eines dynamischen Szenarios wird in dieser Evaluation das Löschen eines Substratservers als dynamisches Ereignis näher betrachtet. Dies resultiert aus den folgenden Überlegungen heraus:

- Basierend auf den vorgestellten Lösungsstrategien für dynamische VNE-Probleme in Abschnitt 2.2.3 hat kein Algorithmus das Löschen eines Substratservers unterstützt. Somit soll in dieser Evaluation demonstriert werden, dass iMdVNE weitergehende Möglichkeiten als bereits veröffentlichte Algorithmen besitzt und somit neue Anwendungsszenarien ermöglicht.
- Das Löschen eines Substratservers stellt ein komplexes Ereignis dar, insbesondere, wenn auf dem Substratserver schon virtuelle Server platziert sind. Somit muss eine Migration dieser virtuellen Server durchgeführt werden.

- Durch die verringerte Anzahl an Substratservern wird auch die Platzierung der virtuellen Netzwerke eine herausfordernde Aufgabe und es können sich Unterschiede zwischen den beiden Migrationsstrategien (notwendige und optimierende Migration) zeigen.
- Damit Tendenzen ersichtlich sind und fundierte Aussagen getroffen werden können, wird das Ereignis mehrere Male hintereinander ausgeführt.

Aus diesen Gründen werden nach dem Einbetten der virtuellen Netzwerke 20 zufällig ausgewählte Substratserver nacheinander gelöscht. Dabei werden Substratserver mit existierenden Einbettungen, d.h. es laufen virtuelle Server auf diesen Substratservern, bevorzugt, falls dies möglich ist. Somit ist eine Migration dieser virtuellen Server notwendig, um die Anforderungen wieder sicherzustellen. In allen Diagrammen werden daher 60 Ereignisse auf der y-Achse angezeigt, wobei die ersten 40 Ereignisse das Hinzufügen der virtuellen Netzwerke und die letzten 20 Ereignisse das Löschen der Substratserver repräsentieren. Als Vergleichsalgorithmus kommt ein manuell programmierter ILP-basierter Ansatz zum Einsatz (kurz Basis-Ansatz), der die ILP-Formulierungen generiert und zur Laufzeit inkrementell aktualisiert. Zudem wird weiterhin Gurobi verwendet, der inkrementelle Änderungen an der ILP-Formulierung unterstützt.

6.3.2 *FF A.1 - Skalierbarkeit*

> Welchen Einfluss hat die *Modellgröße* auf die Laufzeit zur Lösung des VNE-Problems?

Bei dieser Forschungsfrage wird die Skalierbarkeit von iMdVNE im Vergleich zum inkrementell arbeitenden Basis-Ansatz zur Lösung eines VNE-Problems in einem kleinen oder großen Rechenzentrum untersucht und diskutiert. Dazu werden wir die beiden Migrationsstrategien (notwendig und optimierend) getrennt voneinander betrachten.

Notwendige Migration Wir beginnen mit der notwendigen Migration. Die Ergebnisse für diese Migrationsstrategie sind in Abb. 6.9 zu finden, wobei die Laufzeit zur Lösung des VNE-Problems in einer logarithmischen Skalierung über der Anzahl an durchgeführten Ereignissen dargestellt wird. Die gestrichelten Linien zeigen dabei den Basis-Ansatz und die durchgezogenen Linien die Datenpunkte von iMdVNE jeweils für das kleine (kSN) und das große Substratnetzwerk (gSN). Bei den ersten 40 Ereignissen werden einzelne virtuelle Netzwerke eingebettet (VN neu) und in den letzten 20 Ereignissen Substratserver gelöscht (Lösche Sr.). Beim Löschen eines Substratservers wird, falls möglich, ein Server mit eingebetteten virtuellen Servern ausgewählt und entfernt. In Abb. 6.10 sind die Summen über alle gruppierten Ereignisse dargestellt. Neben der reinen Laufzeit zur Lösung der ILP-Formulierung werden auch der MT-Anteil bei iMdVNE bzw. die sonstigen Anteile für den Basis-Ansatz visualisiert. Die Gesamtlaufzeit zur Einbettung der virtuellen Netzwerke bzw. der Migrationen ergibt sich aus der Summe dieser beiden Anteile.

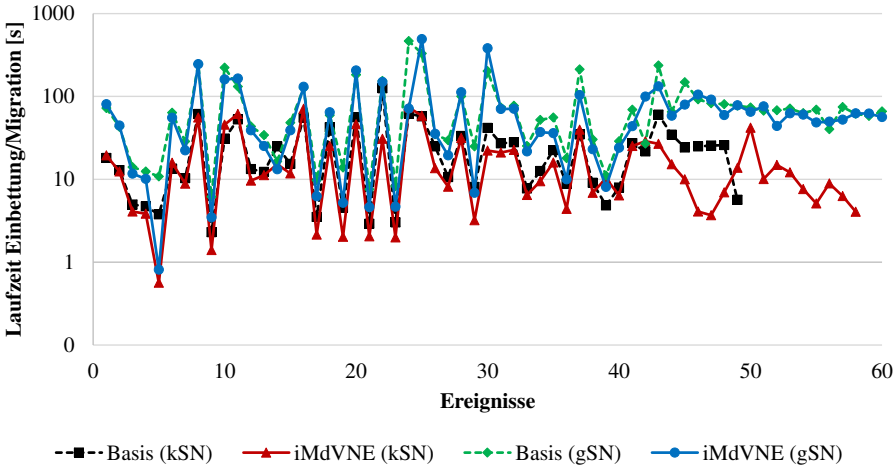


Abbildung 6.9: Vergleich für ein kleines und ein großes Substratnetzwerk mit allen Anforderungen und der *notwendigen Migration*

In Abb. 6.9 wird ersichtlich, dass die Gesamtlaufzeit über die gesamte Anzahl an Ereignissen starken Schwankungen unterliegt, was auf die unterschiedlichen Rahmenbedingungen (z.B. Verteilung eingebetteter virtueller Server oder noch verfügbarer Substratserver) zurückzuführen ist. Da die Kurven von iMdVNE und dem Basis-Ansatz sowohl für das kleine als auch das große Netzwerk sehr ähnlich verlaufen, ist dies ein Indiz dafür, dass die generierten ILP-Formulierungen einen ähnlichen Aufbau und Schwierigkeitsgrad zum Lösen besitzen. Dies ist natürlich auch auf die vergleichbaren strukturellen Netzwerke und die möglichen Platzierungskandidaten zurückzuführen. Erst ab dem 40. Ereignis für das große Netzwerk verringern sich diese Schwankungen signifikant, da für die Migration der nicht mehr eingebetteten virtuellen Server im Gegensatz zum kleinen Netzwerk eine Vielzahl von kompakten (einfachen) Platzierungen (z.B. in einem Rack oder auf einem Server) möglich ist. Im kleinen Netzwerk kann dieses Verhalten nicht in diesem Maße beobachtet werden, da nur wenige gültige Einbettungen für die reduzierte Menge an Substratservern möglich sind. Da die interne Arbeitsweise des ILP-Lösers für die Abarbeitung von inkrementellen Änderungen nicht ausreichend dokumentiert ist, kann es sein, dass dieser das gesamte VNE-Problem erneut lösen muss und nicht auf vorherige Teilergebnisse zurückgreifen kann. Dadurch lassen sich auch die Schwankungen in der Laufzeit erklären. Zusätzlich fällt auf, dass der Basis-Ansatz nur 10 Substratserver löschen kann (bis zum 50. Ereignis), wo hingegen es iMdVNE möglich ist 9 Substratserver mehr (bis zum 59. Ereignis) zu löschen. Dieses Phänomen lässt sich durch die in der Vergangenheit zufällig ausgewählten Lösungen erklären, welche die nachfolgenden Einbettungen beeinflussen.

Insgesamt wird besonders aus Abb. 6.10 ersichtlich, dass iMdVNE für das Lösen des VNE-Problems zwischen 5 % und 46 % weniger Zeit benötigt als der Basis-Ansatz. Da im Vergleich zum statischen Szenario nicht jedes Mal das Substratnetzwerk neu aufgebaut wird und nur die Modelländerungen berücksichtigt werden müssen, kann die Mustererkennung effizienter auf die Modelländerungen reagieren. Dies zeigt sich auch bei der Betrachtung des ILP-Anteils bei iMd-

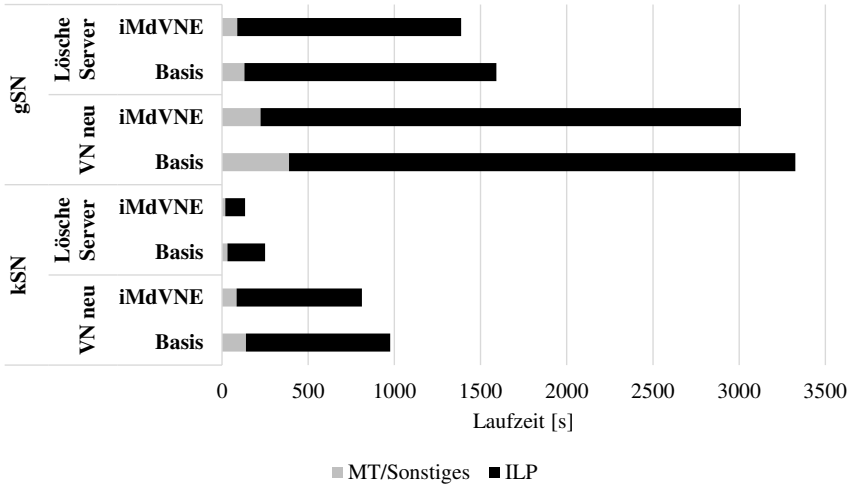


Abbildung 6.10: Summe über die Laufzeit für alle Einbettungen mit einer *notwendigen Migration* aus Abb. 6.9

VNE, da dieser mit 90 % bis 92 % die dominierende Größe darstellt und somit der MT-Anteil nur noch eine untergeordnete Rolle einnimmt. Da iMdVNE auch eine verringerte ILP-Laufzeit im Vergleich zum Basis-Ansatz aufweist, zeigt sich hier die Tendenz, dass durch die Nutzung der MT-Technologien die Anzahl an möglichen Platzierungskandidaten (bzw. ILP-Variablen) reduziert werden kann. Vergleicht man nun den MT-Anteil und den Anteil des Basis-Ansatzes zur Detektion der Modelländerungen und Aktualisierung der ILP-Formulierung fällt auf, dass sich dieser in einem vergleichbaren zeitlichen Rahmen bewegt, wobei der MT-Anteil hierbei generell etwas niedriger ausfällt. Das lässt sich auf die Nutzung der inkrementellen MT-Technologien zurückführen, welche mit effizienteren Methoden Änderungen finden und weiterleiten können als die handgeschriebene Implementierung im Basis-Ansatz.

Bei der Betrachtung der Modellgröße fällt auf, dass iMdVNE für das Einbetten eines neuen Netzwerkes (Löschen eines Substratservers) 812 s (134 s) für kSN und 3012 s (1387 s) für gSN benötigt. Somit hat die Vergrößerung des Substratnetzwerks um 50 % (von 80 auf 120 Substratserver) die Auswirkung, dass für das Hinzufügen eines neuen virtuellen Netzwerks bei iMdVNE (dem Basis-Ansatz) ca. 4-mal (ca. 3-mal) mehr Zeit benötigt wird, was vergleichbar ist zum statischen Szenario. Für das Löschen eines Substratservers wird dabei sogar 10-mal (ca. 6-mal) mehr Zeit aufgewendet. Diese exponentielle Tendenz bei der Laufzeit kann dadurch erklärt werden, dass die Laufzeit zur Lösung der ILP-Formulierung exponentiell mit der Anzahl an Variablen wächst, die sich proportional zu der Anzahl an virtuellen Elementen, multipliziert mit den Substratelementen, erhöht ($|\text{VN}| * |\text{SN}|$).

Optimierende Migration Die Ergebnisse für die optimierende Migration sind in Abb. 6.11 und Abb. 6.12 zu finden. Dabei wird nach jedem Ereignis, unabhängig davon, ob eine Anforderung verletzt wurde oder nicht, berechnet, ob eine bessere Lösung als die aktuelle vorliegende Einbettung vorhanden ist, um Optimalität sicherzustellen. Auch hier wird in Abb. 6.11 die Laufzeit zur Lösung

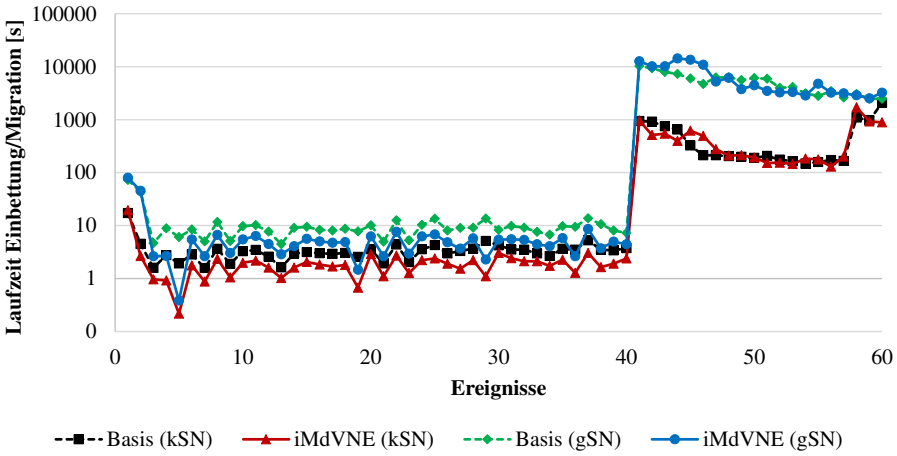


Abbildung 6.11: Vergleich für ein kleines und ein großes Substratnetzwerk mit allen Anforderungen und der *optimierenden Migration*

des VNE-Problems über die Anzahl der durchgeführten Ereignisse aufgezeigt. In Abb. 6.12 sind die Summen über alle gruppierten Ereignisse in logarithmischer Skalierung dargestellt, wobei neben dem MT- bzw. sonstigen Anteil zur Lösung des VNE-Problems auch der ILP-Anteil des ILP-Lösers dargestellt wird.

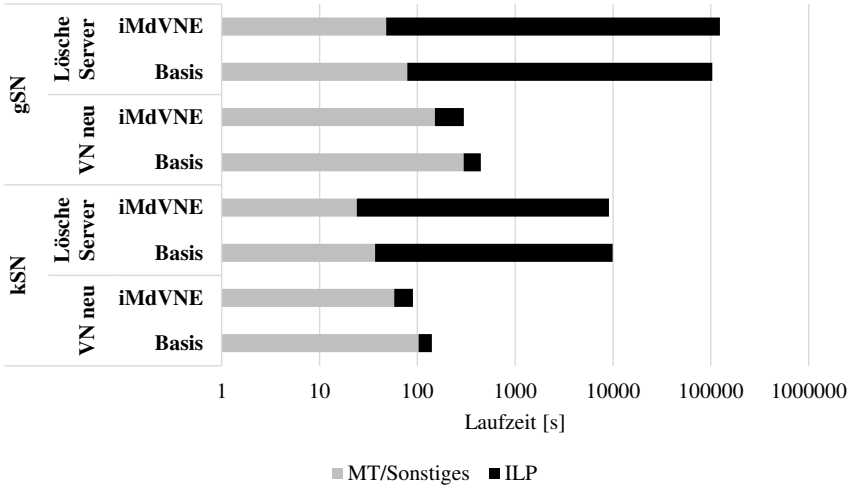


Abbildung 6.12: Summe über die Laufzeit für alle Einbettungen mit einer *optimierenden Migration* aus Abb. 6.11

Anhand von Abb. 6.11 erkennt man einen deutlichen Anstieg der Laufzeit ab dem 40. Ereignis, also ab dem Löschen der Substratserver und dem damit einhergehenden Löschen der Platzierungsvariablen bzw. ILP-Variablen. Da durch das Löschen von ILP-Variablen der ILP-Löser (Gurobi) oftmals die vorher intern berechneten Teilergebnisse verwirft und das VNE-Problem komplett neu berechnet [8], muss nun ein VNE-Problem mit einer Batchgröße von 40, d.h. allen bisher eingebetteten virtuellen Netzwerken, gelöst werden. Hier macht sich auch der Laufzeitunterschied zwischen einem großen und einem kleinen Substratnetzwerk wesentlich stärker bemerkbar, da hier die resultierenden ILP-Formulierungen eine signifikant unterschiedliche Anzahl von ILP-Variablen und -Ungleichungen auf-

weisen und die Laufzeit zum Lösen der ILP-Formulierung exponentiell mit der Anzahl an Variablen steigt. In Abb. 6.10 wird dies beim Vergleich der Summen zwischen dem kleinen und großen Netzwerk beim Löschen eines Substratservers (Lösche Sr.) noch einmal deutlich. Beim Vergleich des ILP-Anteils von iMdVNE und dem Basis-Ansatz fällt auf, dass dieser mit 36 % bei iMdVNE (26 % beim Basis-Ansatz) beim Hinzufügen eines Substratnetzwerkes einen kleinen Anteil gegenüber dem MT-Anteil bzw. der Verarbeitung von Modelländerungen ausmacht. Dies stellt im Gegensatz zum Löschen des Substratservers, wo der MT-Anteil wieder vernachlässigt werden kann, einen Unterschied zur notwendige Migration dar. Auch fällt auf, dass die Summe über alle Einbettungen von virtuellen Servern bei der notwendigen Migration fast um einen Faktor 10 höher ist als bei der optimierenden Migration. Dies kann folgendermaßen erklärt werden: Bei der notwendigen Migration werden vorhandene ILP-Variablen je nach aktueller Einbettung auf null oder eins festgesetzt, um eine Migration zu verhindern. Bei Untersuchungen bezüglich dem inkrementellen Festsetzen von ILP-Variablen wurde festgestellt, dass Gurobi hierbei mehr Zeit zum Aktualisieren der internen ILP-Problemstellung benötigt [8], wodurch sich die Laufzeit zum Lösen des VNE-Problems erhöht. Bei der optimierenden Migration hingegen werden nur neue ILP-Variablen und -Ungleichungen hinzugefügt, wodurch der ILP-Löser intern vorhandene (Teil-)Ergebnisse wiederverwenden kann und die Berechnung somit beschleunigt wird.

Bei Betrachtung der Modellgröße fällt auf, dass bei Vergrößerung des Substratnetzwerkes iMdVNE ca. 3-mal mehr Zeit beim Hinzufügen eines virtuellen Netzwerkes und ca. 14-mal mehr Zeit beim Löschen eines Substratservers benötigt. Beim Basis-Ansatz liegen diese Werte in einem ähnlichen Bereich mit einer Erhöhung der Zeit um einen Faktor von 4 beim Hinzufügen eines virtuellen Netzwerkes und einer Erhöhung um einen Faktor von 10 beim Löschen eines Substratservers. Damit zeigt sich auch hier eine exponentielle Tendenz für die Laufzeit zum Lösen des VNE-Problems mit vergleichbaren Faktoren wie bei der notwendigen Migration.

Insgesamt erkennt man im Diagramm und besonders in der Tabelle, dass iMdVNE in fast allen Fällen eine geringere oder vergleichbare Gesamtlaufzeit aufweist. Beim kleinen Netzwerk benötigte iMdVNE ca. 36 % weniger Laufzeit beim Hinzufügen eines virtuellen Netzwerkes im Vergleich zum Basis-Ansatz. Beim Löschen eines Substratservers in einem großen Netzwerk lag iMdVNE nur ca. 19 % über der Zeit des Basis-Ansatzes. Somit sind das Verhalten und die Werte ähnlich zu der notwendigen Migration.

Fazit Die Forschungsfrage FF A.1 zur Untersuchung der Skalierbarkeit beantworten wir daher wie folgt: Bei Erhöhung der Modellgröße von einem kleinen zu einem großen Substratnetzwerk erhöhte sich die Laufzeit zur Einbettung eines neuen virtuellen Netzwerkes, sowohl für die notwendige als auch für die optimierende Migration, ungefähr um den Faktor 3. Beim Löschen eines Substratservers hingegen erhöhte sich die Laufzeit bei iMdVNE im Falle der optimierenden Migration um den Faktor 14 (Basis-Ansatz um den Faktor 10) und für die notwendi-

ge Migration um den Faktor 10 (Basis-Ansatz um den Faktor 6). Die Modellgröße hat somit einen erheblichen Einfluss auf die Gesamt- und ILP-Laufzeit.

Bei der Gesamtlaufzeit lag iMdVNE für die notwendige Migration in den meisten Fällen unter dem Basis-Ansatz. Im Falle des Löschens eines Substratservers in einem kleinen Netzwerk sogar um ca. 46 % darunter. Bei der optimierenden Migration konnten die Laufzeiten zur Einbettung eines neuen virtuellen Netzwerkes in einem kleinen Netzwerk durch iMdVNE um bis zu 36 % verringert werden, wobei sie sich beim Löschen eines Substratservers im großen Netzwerk durch iMdVNE um ca. 19 % gegenüber dem Basis-Ansatz erhöhte. Somit konnte für iMdVNE in diesem Anwendungsszenario nachgewiesen werden, dass durch die Spezifikation des VNE-Problems auf einem hohen Abstraktionsniveau ein vergleichbarer, in Teilen sogar effizienterer Algorithmus, unter Gewährleistung der Korrektheit und Optimalität, als ein handgeschriebenes ILP-basiertes Programm abgleitet werden kann.

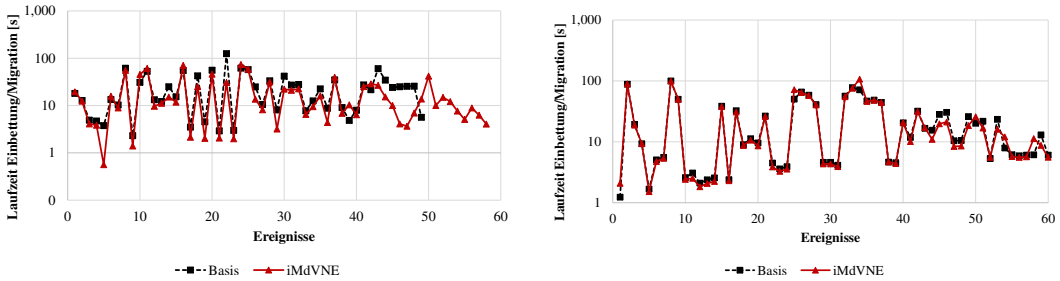
6.3.3 *FF A.2 - Anforderungen*

> Hat die *Anzahl an Anforderungen* einen Einfluss auf die Laufzeit zur Lösung des VNE-Problems?

Bei dieser Forschungsfrage wird untersucht, wie sich die Anzahl an (einschränkenden) Anforderungen auf die Laufzeit zur Lösung des VNE-Problems für das kleine Substratnetzwerk auswirkt. Dazu wird das VNE-Problem im ersten Fall mit allen Anforderungen (AA) und im zweiten Fall nur mit den Basis-Anforderungen (BA) gelöst.

Notwendige Migration In Abb. 6.13 sind die Ergebnisse der Messreihen für die notwendige Migration dargestellt. In den beiden Diagrammen werden die Laufzeiten zur Lösung des VNE-Problems logarithmisch über die durchgeführten Ereignisse abgebildet, wobei die schwarze gestrichelte Linie den Basis-Ansatz und die rote durchgezogene Linie iMdVNE repräsentiert. Zusätzlich sind in Abb. 6.14 die Summen für die Ereignisse mit den jeweiligen Anforderungen (AA oder BA) dargestellt.

In den beiden Diagrammen in Abb. 6.13 wird deutlich, dass sich iMdVNE und der Basis-Ansatz sehr ähnlich verhalten. Lediglich bei der Sicherstellung von allen Anforderungen liegt iMdVNE ab dem 42. Ereignis für die nächsten 6 Ereignisse deutlich unter dem Basis-Ansatz. Außerdem kann der Basis-Ansatz ab dem 49. Ereignis keine Lösung mehr finden, wobei iMdVNE noch bis zum 58. Ereignis das VNE-Problem erfolgreich lösen kann. Diese beiden Beobachtungen können durch vorher getroffene Einbettungsentscheidungen erklärt werden, wie dies auch schon in anderen Konstellationen festgestellt wurde. Anhand der Daten aus Abb. 6.14 wird ersichtlich, dass sich durch eine Erhöhung der Anforderungen die Zeit zum Einbetten eines neuen virtuellen Netzwerkes von 1080 s auf 812 s (ca. 25 %) und beim Löschen eines Substratservers sogar um ca. 50 % (von 265 s auf 134 s) verringert. Dieses Verhalten lässt sich auf den MT-Teil von iMdVNE zurückführen, da sich durch die vergrößerte Anzahl an Anforderungen die Menge an möglichen Platzierungskandidaten verringert und damit weniger potenzielle Möglichkeiten



(a) Alle Anforderungen **(b)** Basis-Anforderungen

Abbildung 6.13: Laufzeit der Einbettung und Migration über die Anzahl der bearbeiteten virtuellen Netzwerke für das kleine Szenario und einer *notwendigen Migration*

durch den ILP-Löser betrachtet werden müssen. Da beim Löschen eines Substratservers nur die nicht mehr eingebetteten virtuellen Server beachtet werden, muss der MT-Teil nur die betreffenden Übereinstimmungen aktualisieren und an den ILP-Löser weiterreichen. Durch die zusätzlichen Anforderungen wird dabei die Menge an möglichen Platzierungen weiter eingeschränkt, wodurch der ILP-Löser schneller eine Lösung finden kann. Hierbei unterscheidet sich das Verhalten zum statischen Anwendungsfall, da sich dort die Laufzeit durch die zusätzlichen Anforderungen erhöht hat. Da im statischen Fall die ganze Problemstellung erneut berechnen muss, zeigt sich hier der Vorteil der Nutzung von inkrementellen Änderungen in der ILP-Formulierung, da der ILP-Löser im besten Fall nur die zusätzlichen Änderungen betrachten muss. Bei der Untersuchung der Gesamtlaufzeit wird deutlich, dass iMdVNE zum Lösen des VNE-Problems im besten Fall 46 % weniger und im schlechtesten Fall nur 4 % mehr Zeit als der Basis-Ansatz benötigt hat.

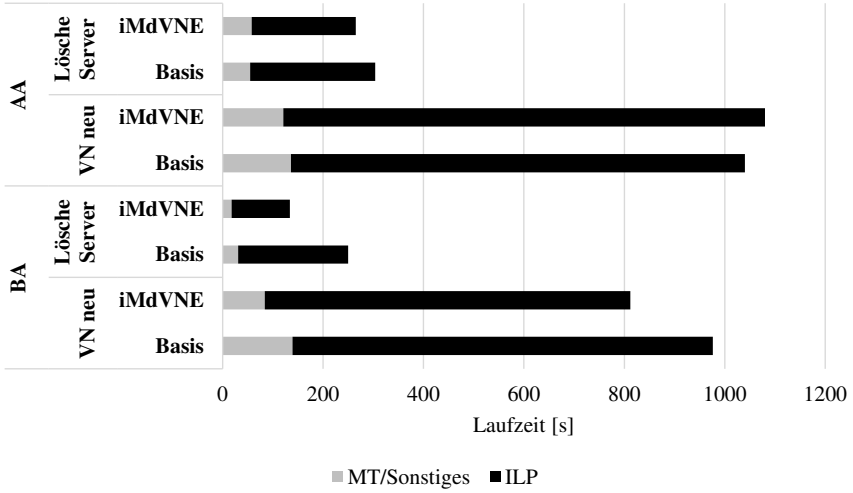
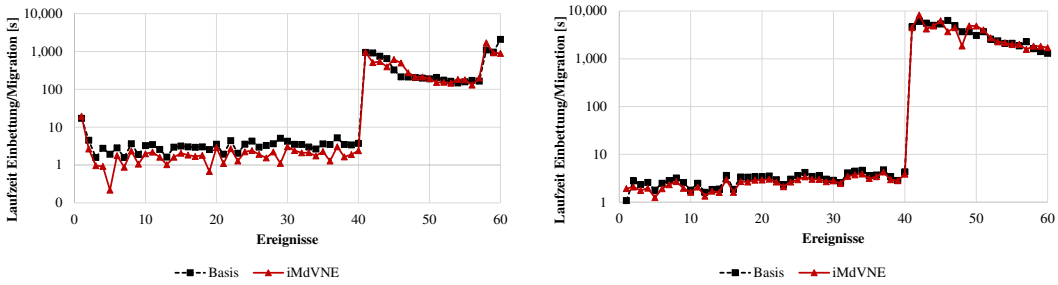


Abbildung 6.14: Summe über die Laufzeit für alle Einbettungen in einem kleinen Substratnetzwerk mit einer *notwendigen Migration* aus Abb. 6.13

Optimierende Migration Die Ergebnisse für die optimierende Migration sind in den beiden Diagrammen in Abb. 6.15 zu finden. Auch hier wird, wie zu-



(a) Alle Anforderungen **(b)** Basis-Anforderungen

Abbildung 6.15: Laufzeit der Einbettung und Migration über die Anzahl der bearbeiteten virtuellen Netzwerke für das kleine Szenario und einer *optimierenden Migration*

vor, die Laufzeit zum Lösen des VNE-Problems logarithmisch über die durchgeführten Ereignisse für iMdVNE und dem Basis-Ansatz dargestellt. In Abb. 6.15a werden dabei alle Anforderungen und in Abb. 6.15b die Basis-Anforderungen berücksichtigt. In Abb. 6.16 sind die Summen für diese beiden Diagramme nach Ereignissen und den Anforderungen zusammengefasst dargestellt, wobei eine logarithmische Skalierung verwendet wird.

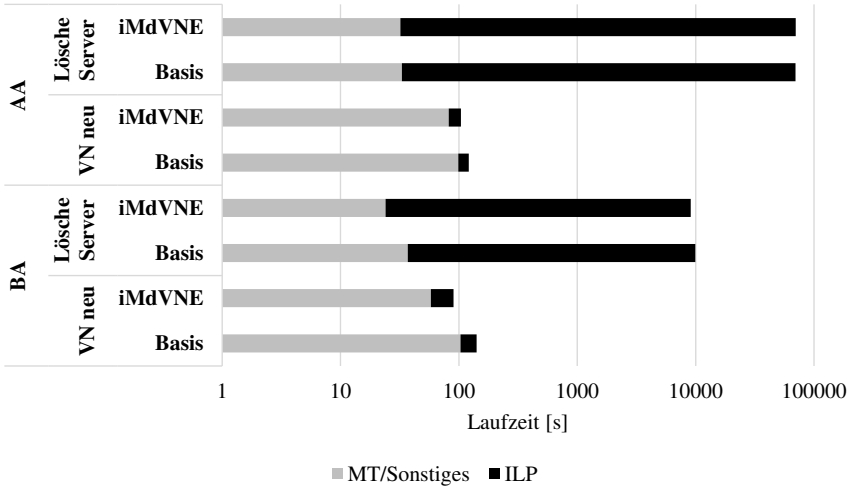


Abbildung 6.16: Summe über die Laufzeit für alle Einbettungen in einem kleinen Substratnetzwerk mit einer *optimierenden Migration* aus Abb. 6.15

In beiden Diagrammen aus Abb. 6.15 erkennt man, dass sich iMdVNE und der Basis-Ansatz sehr ähnlich verhalten, da die Kurven ungefähr parallel zueinander verlaufen oder sogar fast deckungsgleich sind. Allerdings kann man neben kleineren Unterschieden ab dem 40. Ereignis, d.h. dem Löschen eines Substratservers, auch einen Laufzeitunterschied zwischen dem 1. und dem 40. Ereignis bei der Betrachtung von allen Anforderungen feststellen. In diesem Bereich erkennt man deutlich, dass iMdVNE bei allen Ereignissen weniger Laufzeit zum Lösen des VNE-Problems benötigt, wobei hierbei die logarithmische Skalierung beachtet werden muss. Bei Betrachtung der Daten aus Abb. 6.16 wird ersichtlich, dass sich durch die Erhöhung der Anforderungen iMdVNE die Gesamtlaufzeit zur Lösung der Problemstellung um ca. 15 % verringern kann, wobei sich der

ILP-Anteil um ca. 45 % erhöht. Hier zeigt sich deutlich der Effekt, dass die inkrementellen MT-Technologien durch die zusätzlichen Anforderungen effizienter die Anzahl an Übereinstimmungen und damit auch den Platzierungskandidaten aktualisieren können, wohingegen der ILP-Löser mehr Zeit zum Lösen dieser Problemstellung benötigt. Beim Basis-Ansatz ist eine Verringerung der Laufzeit nicht zu beobachten, da sowohl für die handgeschriebene Implementierung zur Bearbeitung der zusätzlichen Anforderungen als auch für die Berechnung der Lösung durch den ILP-Löser mehr Zeit benötigt wird. Insgesamt kann auch hier iMdVNE im Vergleich zum Basis-Ansatz das VNE-Problem bis zu 36 % schneller lösen und benötigt maximal dieselbe Laufzeit wie der Basis-Ansatz.

Fazit Die Forschungsfrage FF A.2 zur Untersuchung der Laufzeit in Bezug auf die Anzahl an Anforderungen beantworten wir daher wie folgt: Bei Betrachtung der notwendigen und optimierenden Migration hat eine Erhöhung der Anzahl an Anforderungen eine Verringerung der Gesamtlaufzeit sowohl für das Einbetten eines neuen virtuellen Netzwerks als auch das Löschen eines Substratservers zur Folge gehabt. Dabei konnte die Laufzeit im Falle des Löschens eines Substratservers bei der notwendigen Migration teilweise um bis zur Hälfte reduziert werden. Zusätzlich wurde deutlich, dass iMdVNE im Vergleich zum Basis-Ansatz ein vergleichbares Laufzeitverhalten aufweist, wobei Tendenzen zu einer signifikanten Reduzierung der Laufzeit festgestellt wurden.

6.3.4 *FF B.1 - Korrektheit und Optimalität*

> In welchem Umfang wird die *Korrektheit und Optimalität* bei iMdVNE sichergestellt?

Diese Forschungsfrage, welche Methoden und Techniken verwendet wurden, um die Korrektheit und Optimalität der Ergebnisse sicherzustellen, lässt sich analog zu den beiden Forschungsfragen FF B.1 in Abschnitt 6.2.5 und FF B.2 in Abschnitt 6.2.6 aus dem statischen Anwendungsszenario beantworten. In Bezug auf die Korrektheit wurden dieselben Methoden und Technologien angewendet wie beim statischen Szenario. Somit wurden die Konstruktionsmethodik aus Abschnitt 4.3, ein Vergleich der Ergebnisse von iMdVNE mit dem Basis-Ansatz, eine Konsistenzprüfung nach der Einbettung und Migration von virtuellen Elementen und Unit-Tests genutzt. Durch den Einsatz dieser Methoden und Techniken kann die Wahrscheinlichkeit für inkorrekte Ergebnisse stark verringert werden.

Die Sicherstellung der Optimalität ist nur bedingt durch einen Vergleich zwischen den Werten der Zielfunktion und der Einbettungen im Substratnetzwerk von iMdVNE und dem Basis-Ansatz zu erreichen, da durch die Nutzung der inkrementellen Techniken, die ILP-Formulierungen nicht mehr vergleichbar sind. Da die ursprüngliche ILP-Formulierung durch jedes Ereignis verändert und nicht wie beim statischen Szenario jedes Mal von Grund auf neu generiert wird, entstehen unterschiedliche ILP-Formulierungen, die untereinander nicht mehr oder nur sehr bedingt verglichen werden können. Daher wird die Wahrscheinlichkeit zum Erhalten einer optimalen Lösung durch Unit-Tests und stichprobenartige Untersuchungen der Lösungen erhöht.

6.3.5 *Gefährdung der Validität der Ergebnisse*

Bezüglich der Validität der Ergebnisse wird auf die Anmerkungen im statischen Szenario (siehe Abschnitt 6.2.7) verwiesen, da die Softwareumgebung und die verwendeten Technologien vergleichbar zu MdVNE sind. Lediglich die Aktualisierung der ILP-Formulierung durch die inkrementellen Änderungen stellt eine zusätzliche Fehlerquelle dar, da alle auftretenden Ereignisse, deren Auswirkungen und Ergebnisse beachtet und integriert werden müssen. Durch systematische Tests (insgesamt 225 Unit-Tests) werden manuell berechnete Szenarien mit den tatsächlich entstandenen Ergebnissen verglichen.

Das aktuelle verwendete dynamische Anwendungsszenario für diese Evaluation dient eher der grundlegenden Darstellung der Möglichkeiten zur Realisierung von verschiedenen Ereignissen und dem Aufzeigen der damit verbundenen Laufzeiten zur Lösung des VNE-Problems. Obwohl die verwendeten Daten für die virtuellen Netzwerke und dem Substratnetzwerk sich an realen Umgebungen und anderen Evaluationen bestehender Veröffentlichungen orientieren, ist die Abfolge an Ereignissen nicht sehr praxisnah. So werden in der Realität verschiedene Ereignisse in unterschiedlichen Abfolgen und Häufigkeiten eintreffen, wodurch sich ein anderes Verhalten für iMdVNE, aber auch dem ILP-basierten Basis-Ansatz ergeben. Damit die Evaluation des dynamischen Szenarios aber vergleichbar zum statischen Szenario ist und um die Möglichkeiten von iMdVNE aufzuzeigen, wurde der vorliegende Aufbau für diese Evaluation gewählt. Davon unberührt bleibt die zugrundeliegende Problemstellung, die maßgeblich durch die Topologien der Netzwerke und den spezifizierten Anforderungen bestimmt wird. Diese Problemstellung bzw. die gewählte 2-Tier-Topologie mit den verwendeten Anforderungen ist ein repräsentatives Beispiel für eine große Klasse von VNE-Problemen und kann daher aussagekräftige Tendenzen und Ergebnisse für das Verhalten von VNE-Algorithmen liefern.

Da die Unterstützung von inkrementellen Änderungen für ILP-Löser (z.B. Gurobi oder Cplex) nur sehr rudimentär dokumentiert ist, kann es sein, dass ein anderer ILP-Löser (z.B. Cplex) für konkrete Problemstellungen bessere Laufzeiten aufweist. Beispielsweise konnte bei weiteren Untersuchungen gezeigt werden, dass Gurobi (scheinbar) interne Ergebnisse durch das Löschen von ILP-Variablen (z.B. dem Löschen eines Substratservers) verwirft und neu beginnt [8]. Bei Cplex wird durch einen Reparaturalgorithmus versucht das Löschen in die bestehende ILP-Formulierung zu integrieren, was im schlechtesten Fall zu einer Laufzeiterhöhung führen kann. Da bei [8] gezeigt wurde, dass Gurobi inkrementelle Änderungen an der ILP-Formulierung tendenziell schneller umsetzt und auch andere Veröffentlichungen z.B. [105, 78, 63, 118] diesen ILP-Löser verwenden, fiel die Wahl für diese Evaluation auf Gurobi.

6.3.6 *Zusammenfassung*

In dieser Evaluation für ein dynamisches Anwendungsszenario konnten wir zeigen, dass der vorgestellte iMdVNE-Ansatz ähnlich zu einem manuell erstellten ILP-basierten Ansatz skaliert und in einigen Fällen sogar nur ungefähr die Hälfte

der Laufzeit benötigte. Dabei zeigte sich vor allem beim großen Substratnetzwerk oder der notwendigen Migration, dass der MT-Anteil eine untergeordnete Rolle spielt und die Zeit des ILP-Löser den dominierenden Faktor darstellt, wobei in den meisten Fällen auch der ILP-Anteil im Vergleich zum Basis-Ansatz verringert werden konnte. Weiter ließ sich in dieser Evaluation aufzeigen, dass es möglich ist komplexe Ereignisse (Löschen eines Substratservers) mit Migration erfolgreich in iMdVNE umzusetzen und dabei in vielen Fällen schneller eine Lösung als ein vergleichbarer ILP-basierter Ansatz zu erhalten. Somit kann ausgehend von einer abstrakten modellbasierten Problemdefinition über eine per-Konstruktion-korrekten Methodik und Technologien zur teilweisen Automatisierung von ausführbarem Quellcode ein effizienter Algorithmus abgeleitet werden, welcher vergleichbar und teilweise sogar effizienter als ein handgeschriebener ILP-basierter Algorithmus ist. Zusätzlich kann durch den modellbasierten Ansatz die Korrektheit und Optimalität sichergestellt, die Anpassbarkeit an neue Anwendungsszenarien erhöht und die Qualität des Quellcodes verbessert werden.

6.4 Vergleich MdVNE und iMdVNE

Zum Abschluss wird noch ein kurzer Vergleich zwischen MdVNE und iMdVNE gezogen, um die Veränderungen und Unterschiede, die sich durch die Integration der inkrementellen MT- und ILP-Technologien ergeben, zu verdeutlichen. Dabei soll kurz auf die unterschiedlichen Methoden und das Laufzeitverhalten basierend auf der vorliegenden Evaluation eingegangen werden.

Der größte Unterschied zwischen MdVNE und iMdVNE besteht bei der Durchführung von Ereignissen, insbesondere von Einbettungen. So wird bei MdVNE für jede Einbettung eines neuen virtuellen Netzwerks das komplette Modell mithilfe von MT-Technologien in eine ILP-Formulierung überführt, wodurch beispielsweise für jede Einbettung das Substratnetzwerk eingelesen, die passenden ILP-Variablen und -Ungleichungen abgeleitet, das VNE-Problem gelöst und danach alle intern vorhandenen (Teil-)Ergebnisse gelöscht werden. Im Gegensatz hierzu wird durch die Integration von inkrementellen Technologien bei iMdVNE das Substratnetzwerk nur einmal initialisiert und danach werden nur noch alle Änderungen am Modell nacheinander bearbeitet. So können interne (Teil-)Ergebnisse nach jeder Einbettung weiterverwendet werden, wobei dies nicht nur für den MT-Anteil der Fall ist, sondern auch für den ILP-Teil möglich ist, soweit dies durch den ILP-Löser unterstützt wird. Somit kann der ILP-Löser interne (Teil-)Ergebnisse weiterverwenden und somit die einzelnen Problemstellungen effizienter lösen.

Beim Vergleich der Laufzeiten von MdVNE und iMdVNE werden diese Einsparungen auch ersichtlich. So benötigt MdVNE für die Einbettung von neuen virtuellen Netzwerken im kleinen Substratnetzwerk (großen Netzwerk) und allen Anforderungen 2223 s (8850 s) und iMdVNE mit der notwendigen Migration 812 s (3011 s) und mit der optimierenden Migration nur 90 s (298 s). Somit kann durch die Nutzung von inkrementellen Technologien die Laufzeit um den Faktor 2 bis 30 verringert werden. Diese Verringerung der Laufzeit spiegelt sich auch im Vergleich zum Basis-Ansatz wider, da MdVNE hier im Vergleich zum Basis-Ansatz noch ungefähr doppelt soviel Zeit benötigte, um die Lösung zu berechnen,

wo hingegen iMdVNE im Vergleich zum Basis-Ansatz teilweise nur noch auf die Hälfte der Zeit angewiesen ist.

Zusammenfassend kann man beobachten, dass die Rechenzeit einer abgeleiteten Implementierung für (i)MdVNE sich vergleichbar zu einer manuellen handoptimierten Implementierung verhält. Zusätzlich kann beim dynamischen VNE-Problem festgestellt werden, dass die Nutzung von inkrementellen MT- und ILP-Technologien die Laufzeit nochmal signifikant verringert und sich eine Verringerung des MT-Anteils im Verglich zum statischen Szenario zeigt.

7

VERWANDTE ARBEITEN

In diesem Kapitel diskutieren wir verwandte Arbeiten zu der in dieser Arbeit vorgestellten Problemstellung, den entwickelten Ansätzen und Lösungsmethoden. Zunächst diskutieren wir Lösungsstrategien für die Einbettung von virtuellen Netzwerken in Rechenzentren und andere modellbasierte Ansätze aus verschiedenen Domänen, die ähnliche Optimierungsprobleme lösen. Danach gehen wir auf Ansätze ein, die auf Kombinationen von ILP mit MT oder anderen suchbasierten Techniken beruhen und diskutieren inkrementelle Ansätze für Modelltransformationen. Zum Abschluss stellen wir weitere Methoden vor, die per-Konstruktion die Korrektheit eines Systems sicherstellen.

7.1 Einbettung virtueller Netzwerke in Rechenzentren

Das Problem der Einbettung von virtuellen Netzwerken in Rechenzentren und die Virtualisierung von Rechenzentrumsnetzwerken wurde in der Literatur schon ausführlich untersucht. Ein Überblick über verschiedene Algorithmen und Ansätze ist in [7, 28] und in Abschnitt 2.2.3 zu finden. Dabei wurden viele Algorithmen entwickelt, um den Suchraum für dieses NP-harte Optimierungsproblem zu reduzieren [2]. Beispielsweise wird bei Guo et al. [41] SecondNet vorgestellt, ein heuristischer Ansatz zur Einbettung von Teilmengen virtualisierter Rechenzentren in baumbasierte Rechenzentren. Dabei wird der Fokus bei den Ressourcen auf die Bandbreite von Verbindungen und den Slots von virtuellen Maschinen gelegt, wodurch der Suchraum reduziert und damit auch die Zeit zum Lösen des VNE-Problems verringert werden kann. Bei Zeng et al. [118] wird zusätzlich der Datenverkehr zwischen einzelnen virtuellen Maschinen mit berücksichtigt und als Optimierungsziel die Minimierung der resultierenden Kommunikationskosten verwendet. Neben einem heuristischen Ansatz für größere Rechenzentren wird zusätzlich auch ein ILP-basierter Ansatz für kleine Rechenzentren vorgestellt. Bei Rabbani et al. [79] werden zusätzlich die Beziehungen zwischen Switches und Verbindungen mit beachtet und als Optimierungsziel neben der Minimierung der Kommunikationskosten auch die Serverfragmentierung minimiert, wodurch Engpässe im Netzwerk vermieden werden sollen.

Im Vergleich zu den erwähnten Algorithmen und Ansätzen unterstützen MdVNE und iMdVNE unterschiedliche Netzwerktopologien, Anforderungen und Optimierungsziele, welche durch das Anpassen des Metamodells (Klassendiagramm und OCL-Zusicherungen), der MT- und ILP-Konfiguration ermöglicht werden. Dabei kann durch die per-Konstruktion-korrekte Methodik sichergestellt werden, dass die gefundenen Lösungen korrekt in Bezug auf die Anforderungen

und optimal in Bezug auf das Optimierungsziel sind. Dies ist bei Verwendung von Heuristiken meist nur schwer nachweisbar. Allerdings haben sowohl die durchgeführte Evaluation als auch andere Veröffentlichungen aufgezeigt, dass ein Ansatz, der optimale Einbettungen findet, nicht skaliert und für die Praxis in großen Rechenzentren wegen der langen Laufzeit zur Lösung der Problemstellung eher ungeeignet ist. Aus diesem Grund sind in der Literatur meist Heuristiken zu finden, um die Zeit zum Lösen zu reduzieren und die Algorithmen in realen Szenarien zu nutzen. Ein korrekter und optimaler Algorithmus kann jedoch als Referenz genutzt werden, um die Qualität der gefundenen Lösungen von einem heuristischen Algorithmus im Vergleich zu einer optimalen Lösung zu bewerten. Auch können durch (i)MdVNE (leicht) Heuristiken realisiert werden, was im Ausblick dieser Arbeit thematisiert wird.

7.2 Modellbasierte Ansätze für Optimierungsprobleme

Eine vielversprechende Methode zur plattform- und problemunabhängigen Entwicklung von Anwendungen stellt die modellbasierte (Software-)Entwicklung dar. So spielt die teilweise automatisierbare Verifikation einer Spezifikation und die Generierung von Quellcode in zahlreichen Anwendungen eine wichtige Rolle. Für das Anwendungsszenario Brake-by-Wire aus der Automobilindustrie stellt beispielsweise Pohlmann et al. [77] einen modellbasierten Allokationsansatz vor, um Softwarekomponenten auf elektrischen Steuergeräten zu verteilen. Dazu wird das Platzierungsproblem in einer OCL-basierten Sprache spezifiziert und danach in eine ILP-Formulierung transformiert, durch welche das Optimierungsproblem gelöst werden kann. Für die Platzierung von Softwareimplementierungen auf Hardwarekomponenten nutzt Götz et al. [34, 35] eine Modell-zu-Text-Transformation, um ein Metamodell, welches die Spezifikation dieser Platzierungen darstellt, in eine ILP-Formulierung zu überführen. Somit kann mithilfe eines ILP-Lösers eine Lösung für dieses Optimierungsproblem berechnet werden. Zur Transformation des Metamodells in eine ILP-Formulierung kommen dabei Reference Attribute Grammars (RAQ) [48] zum Einsatz. Bei Schöne et al. [87] wird dieser Ansatz auch auf inkrementelle Änderungen angewendet, wodurch ein Optimierungsproblem aus dem Bereich selbstadaptiver Systeme gelöst wird. Dabei wird zusätzlich ein Fokus auf die Zwischenspeicherung der (Teil-)Ergebnisse gelegt, um möglichst wenige Attribute und Variablen zu verändern bzw. neu zu berechnen. Im Bereich der Software-definierten Netzwerke stellt Lopes et al. [65] eine Methode zur Erstellung von anwendungs-, controller- und netzwerkunabhängigem Code für Software-definierte Netzwerkanwendungen anhand von modellierten Funktionalitäten und physischen Netzwerken vor. Bei Kluge et al. [55] wird die modellbasierte Softwareentwicklung eingesetzt, um Algorithmen zur Anpassung von Topologien durch Graphtransformationen umzusetzen, wobei sowohl globale als auch lokale Konsistenzbedingungen (z.B. Erhaltung der Konnektivität) eingehalten werden.

Die genannten Ansätze zeigen, dass die modellbasierte Entwicklung eine vielversprechende Methode zur Spezifikation und Ableitung von ausführbaren Anwendungen und Algorithmen in unterschiedlichen Anwendungsdomänen dar-

stellt. Da bisher noch kein modellbasierter Ansatz zur Lösung von VNE-Problemen in Rechenzentren in der Literatur gefunden werden konnte, stellt diese Arbeit eine Methode vor, um ausgehend von einer modellbasierten Problemdefinition ausführbaren Quellcode abzuleiten, welcher korrekte und optimale Lösungen sicherstellt. Dabei werden bereits erfolgreich in anderen Domänen eingesetzte Methoden wie beispielsweise die Ableitung von ILP-Formulierungen aus OCL-basierten Sprachen, Metamodellen und MT-Regeln eingesetzt, um das Optimierungsproblem zu lösen. Auch eine per-Konstruktion-korrekte Methodik analog zu Kluge et al. konnte in dieser Arbeit erfolgreich integriert und an die notwendigen Umgebungsbedingungen angepasst werden.

7.3 Kombination von ILP, MT und suchbasierten Techniken

Die Kombination von Modelltransformationen oder anderen suchbasierten Methoden und Optimierungstechniken wird in unterschiedlichen Domänen und Anwendungsszenarien eingesetzt. Die Veröffentlichung von Zschaler et al. [125] bietet einen guten Überblick über aktuelle Ansätze und Herausforderungen, die sich in diesem Forschungsfeld ergeben. Auch wird ein Prototyp für eine modellbasierte Optimierungstechnik vorgestellt. Bei Strüber et al. [93] wird ein Ansatz und eine Implementierung für die Optimierung eines Modells unter Verwendung einer Fitnessfunktion zur effizienten Erstellung von Mutationsoperatoren für generische Algorithmen präsentiert. Hierbei werden die Mutationsoperatoren mithilfe von Modelltransformationen höherer Ordnung trainiert, um die Leistung und die Qualität zu verbessern. Denil et al. [20] präsentieren einen Ansatz zur Integration von suchbasierten Optimierungstechniken in einen modellgetriebenen Entwicklungsprozess. Dazu werden anhand des Anwendungsfalls zur Erstellung von elektrischen Schaltungen verschiedene Optimierungstechniken wie beispielsweise die randomisierte Suche oder der Bergsteigeralgorithmus zur Lösung des Problems eingesetzt. Einen Ansatz zur Suche einer optimalen Abfolge für Regelanwendungen wird bei Fleck et al. [29] vorgestellt. Hierfür werden sowohl Modelltransformationstechniken als auch suchbasierte Algorithmen eingesetzt. Durch die Auswertung von Fitnesswerten nach jeder (willkürlich durchgeführten) Regelanwendung kann der Ansatz den Suchraum direkt reduzieren, wobei ein globales Optimum nicht garantiert werden kann.

Die vorgestellten Ansätze stellen vielversprechende Kandidaten und Methoden dar, um (i)MdVNE zu erweitern und weiter zu verbessern. So könnte die Integration von weiteren suchbasierten Techniken oder (heuristischen) Algorithmen in den MT-Schritt die Laufzeit weiter verbessern und neue Forschungsfelder zur Lösung von VNE-Problemen eröffnen.

7.4 Inkrementelle Modelltransformationen

Techniken zum Vergleichen von Graph-Mustern und Übereinstimmungen lassen sich in batchbasierte und inkrementelle Mustervergleiche unterteilen. Batch-Lösungen beruhen meist auf Techniken für eine lokale Suche [126] oder dem Bedingungserfüllungsproblem [60], was den Nachteil hat, dass bei einer Änderung

am Modell alle Übereinstimmungen komplett neu gesammelt werden müssen. Im Gegensatz hierzu werden beim inkrementellen Mustervergleich die Änderungen am Modell mit verfolgt, so dass die Menge von neuen, veränderten oder ungültigen Übereinstimmungen effizienter gefunden werden kann. Einige Ansätze zur Umsetzung von inkrementellen Mustervergleichen führen grundlegende Datenstrukturen ein [102] oder nutzen verstärkendes Lernen [51]. Ein weit verbreiteter Ansatz für Modelltransformationen stellt der RETE-Algorithmus [31] dar, welcher alle partiellen Übereinstimmungen überwacht, sodass neue Übereinstimmungen oder Änderungen an Übereinstimmungen effizient gefunden werden können. Damit hängt dieser Ansatz theoretisch nur von der Größe der Modelländerung ab und nicht von der Größe des kompletten Modells. Mehrere Werkzeuge wie z.B. Viatra [100], Democles [101] oder HiPE [26] implementieren Varianten des RETE-Algorithmus, wobei HiPE beispielsweise zusätzlich massiv auf Parallelisierung setzt. Neben dem RETE-Algorithmus existieren weitere vielversprechende Algorithmen wie beispielsweise der Gator-Algorithmus [46], TREAT [70] oder einem Algorithmus präsentiert von Fan et al. [27], wobei die Umsetzung dieser Algorithmen in Werkzeuge zur Mustererkennung noch offene Forschungsfragen aufweisen.

In dieser Arbeit wurden sowohl batchbasierte und inkrementelle Lösungsstrategien für den Mustervergleich eingesetzt. Dabei zeigte sich in der Evaluation, dass inkrementelle Lösungsstrategien einen Laufzeitvorteil aufweisen und somit für weitere Forschungsaktivitäten interessant sind. In der vorliegenden Arbeit und Evaluation wurde Viatra eingesetzt, welches auf dem RETE-Algorithmus aufbaut. Durch den Einsatz von effizienteren Algorithmen und Methoden, wie beispielsweise TREAT oder einer massiven Parallelisierung bei der Suche nach Übereinstimmungen, ergibt sich ein großes Potenzial für Effizienzsteigerungen.

7.5 Garantiert korrekte Konstruktionsmethoden

Im Folgenden stellen wir verwandte Ansätze zur Sicherstellung der Korrektheit eines Systems basierend auf einer per-Konstruktion-korrekten Methodik, Verifikation und Tests vor. Die Grundlagen für Methoden, die per-Konstruktion korrekt sind, legen Heckel et al. [47] in der MT-Domäne. Sie präsentieren Graph-Einschränkungen als Prämisse-Ergebnis-Zusicherungen und stellen eine Konstruktionsmethodik vor, um anhand einer Graph-Einschränkung eine schwächste Vorbedingung aus einer MT-Regel abzuleiten. Diese schwächste Vorbedingung ist notwendig und hinreichend, um die Korrektheit bezüglich der Graph-Einschränkung zu garantieren. In dieser Arbeit stellt daher die Herangehensweise von Heckel et al. die Grundlage für die vorgestellte per-Konstruktion-korrekte Methodik dar. Eine Erweiterung zu diesen Prämisse-Ergebnis-Zusicherungen bilden die verschachtelten Graph-Einschränkungen [43]. Auch die Integration von komplexen Attributeinschränkungen in Prämisse-Ergebnis-Zusicherungen, wie sie in Deckwerth et al. [19] präsentiert wird, kann zusätzliche Möglichkeiten für die Transformation von OCL-Zusicherungen in Prämisse-Ergebnis-Zusicherungen bieten. Dabei stellt die Unterstützung für verschachtelte Graph-Einschränkungen noch ein offenes Forschungsgebiet dar. Zur Unterstützung des Schrittes zur Verfeinerung

des Regelsatzes (siehe Abschnitt 4.3.1.2) wurde mit OCL2AC [74] ein Werkzeug vorgestellt, welches eine vielversprechende Möglichkeit bietet, diesen Schritt zu automatisieren. Hierbei baut OCL2AC auf dem Modelltransformationstool Henshin [4] auf, wobei wir aktuell im Schritt der Kandidatengenerierung das Werkzeug eMoflon bzw. Viatra einsetzen und somit durch eine Anpassung an Henshin dieser Automatisierungsschritt in (i)MdVNE integriert werden könnte. Zur Kodierung von Pfadausdrücken, wie sie für die Substratpfade benötigt wird, stellen die HR*-Graph-Einschränkungen von Radke et al. [80] einen interessanten Ansatz dar. Dabei wird präsentiert, wie die HR*-Graph-Einschränkungen in schwächste Vorbedingungen transformiert werden können.

Verifikationsbasierte Ansätze bewerten die erforderlichen Konsistenzeigenschaften a posteriori [82]. Die Konstruktionsmethodik beispielsweise aus [47] eignet sich für eine statische Verifikation, wodurch die Untersuchung zu Eigenschaften der Korrektheit unabhängig von der Systemgröße ist. Da in vielen Fällen die Einhaltung der Konsistenzeigenschaften oder die Korrektheit nicht unabhängig von der Systemgröße bewiesen werden kann, wird die dynamische Verifikation verwendet, um für eine Teilmenge des gesamten Zustandsraums die Einhaltung der Konsistenzeigenschaften zu gewährleisten. Dabei wird ein Modellprüfer genutzt, um den Zustandsraum bis zu einem gewissen Schwellenwert zu erstellen (z.B. basierend auf einem Zeitbudget oder einer Modellgröße). Danach wird dieser Zustandsraum in Bezug auf die gewünschten Konsistenzeigenschaften hin untersucht. Durch diese Techniken konnten beispielsweise kritische Fehler in den Netzwerkprotokollen SIP und Chord durch Methoden von Zave et al. [116, 117] und Modellprüfern identifiziert werden. Der größte Nachteil bei der dynamischen Verifikation ist allerdings, dass die Korrektheit des untersuchten Systems nur für einen (meist sehr kleinen) Teilraum des gesamten im Allgemeinen unendlich großen Zustandsraum gezeigt werden kann.

Bei *testbasierten Ansätzen* wird ein gegebenes System anhand einer Menge von Eingabedaten ausgeführt und überprüft, ob das resultierende Verhalten (z.B. die Ausgabedaten) den Erwartungen bzw. der Spezifikation entspricht [72]. Ähnlich wie bei der dynamischen Verifikation kann ein testbasierter Ansatz allerdings nicht beweisen, dass ein System korrekt arbeitet. Stattdessen erlaubt es uns, das System in bestimmten (Spezial-)Fällen zu evaluieren und eine Regression zu vermeiden, indem Testfälle aus beispielsweise behobenen Fehlern abgeleitet werden. In dieser Arbeit wurden Unit-Tests eingesetzt, um sicherzustellen, dass sich der aus der TGG-Spezifikation generierte Quellcode in repräsentativen Szenarien wie erwartet verhält.

8

ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde ein neuartiger Ansatz zur Spezifikation und Entwicklung von Algorithmen zur Lösung von VNE-Problemen für Rechenzentren präsentiert. Bei diesen dynamischen Optimierungsproblemen werden virtuelle Netzwerke in Rechenzentren eingebettet, wobei vorher festgelegte Anforderungen eingehalten werden müssen und die gefundenen Lösungen möglich kostengünstig oder hardwareschonend sein sollten. Dabei sind die Problemstellungen sehr vielfältig mit unterschiedlichen Anforderungen oder Zielfunktionen. Auch die Methoden, Herangehensweisen und Algorithmen zum Lösen dieser Problemstellung reichen von optimalen über heuristische oder genetische Algorithmen bis hin zu Techniken aus dem maschinellen Lernen. Durch die unterschiedlichen Problemstellungen, Beschreibungssprachen, Methoden und Technologien, besonders in dynamischen Umgebungen, ist die handgeschriebene Implementierung von VNE-Algorithmen zeitaufwendig, fehleranfällig und untereinander nur schwer vergleichbar. Je nach Ansatz kann die Anpassbarkeit an veränderte Anforderungen oder Zielfunktionen eine herausfordernde Aufgabe sein. Aus diesem Grund präsentieren wir in dieser Arbeit einen modellbasierten Ansatz zur Lösung von VNE-Problemen, der anhand einer ausdrucksstarken modellbasierten Problemdefinition und einer per-Konstruktion-korrekten Methodik die Ableitung einer Lösungsstrategie ermöglicht, die eine Kombination von Modelltransformationen und ganzzahliger linearer Programmierung darstellt.

Im Folgenden fassen wir die präsentierten wissenschaftlichen Beiträge sowie deren Ziele zusammen und beschreiben zukünftige Forschungsfelder, die sich aus dieser Arbeit ergeben.

8.1 Zusammenfassung

In dieser Arbeit wurde zunächst ein Spezifikationsansatz für VNE-Probleme für Rechenzentren vorgestellt, um Problemstellungen auf einem hohen Abstraktionsgrad zu beschreiben. Dieser Spezifikationsansatz, auch modellbasierte Problemdefinition genannt, basiert auf einem (UML-)Klassendiagramm mit OCL-Zusicherungen und einer in OCL formulierten Zielfunktion (**WB 1**). Durch diese modellbasierte Problemdefinition lassen sich vielfältige VNE-Probleme aus der Netzwerkdomäne spezifizieren, anpassen und miteinander vergleichen. Dabei konnten wir alle in der Literatur etablierten Arten von Anforderungen für VNE-Probleme integrieren und anhand eines durchgängigen Beispiels den Bezug zu einer konkreten Problemstellung herstellen. Diese Spezifikation kann dabei als erster Entwicklungsschritt dienen, um eine weitgehend systematische und

(teil-)automatisierbare Erstellung von Algorithmen für VNE-Probleme durchzuführen. Zusätzlich können weitere Analysetechniken wie beispielsweise statische Analysen für diese Problemstellung genutzt werden, um Fehlkonfigurationen, Widersprüche oder Anomalien zu erkennen. Auch lassen sich daraus direkt Tests ableiten, um das Verhalten von gängigen oder extremen Situationen zu verifizieren.

Damit die Entwicklung von VNE-Algorithmen durchgängig mit modellbasierten Methoden und Techniken erfolgen kann, wurde in dieser Arbeit ein modellbasierter Ansatz zur Entwicklung von Algorithmen und zur Lösung von VNE-Problemen vorgestellt (**WB 2**). Dabei lag das Ziel auf einer generischen und anpassbaren Methodik, um VNE-Algorithmen zu erstellen und zu evaluieren. Somit lassen sich verschiedene Algorithmen untereinander vergleichen und genauer untersuchen. Dieser modellbasierte Ansatz kombiniert MT- und ILP-Technologien mit dem Ziel eine anpassbare, wiederverwendbare und deklarative konfigurierbare Methode zur Lösung von (VNE-)Optimierungsproblemen zu entwickeln. So nutzen wir MT-Technologien, um den Suchraum bzw. die Menge an möglichen Platzierungskandidaten zu reduzieren, und ILP-Technologien, um korrekte und optimale Lösungen innerhalb dieser Menge zu finden. Dabei lag in dieser Arbeit der Fokus bei der Sicherstellung, dass korrekte und optimale Lösungen für das VNE-Problem gefunden werden. Dieser modellbasierte Ansatz wurde zunächst für statische Anwendungsszenarien als batchbasierte Lösungsstrategie entwickelt (MdVNE) und danach für dynamische Anwendungsszenarien erweitert (iMdVNE). Hierzu wurden inkrementelle MT- und ILP-Technologien eingesetzt, um nur die Änderungen und deren Auswirkungen zu berücksichtigen, wobei die Generierung und Aktualisierung der ILP-Formulierung auf Basis der inkrementellen MT- und ILP-Technologien durchgeführt wird (**WB 4**).

Im nächsten Schritt wurde nun anhand der modellbasierten Problemdefinition eine Konfiguration (MT-Regelsatz und ILP-Formulierung) für den entwickelten modellbasierten Ansatz zur Lösung von VNE-Problemen präsentiert. Dazu wurde eine per-Konstruktion-korrekte Methodik vorgestellt, die ausgehend von der modellbasierten Problemdefinition systematisch einen MT-Regelsatz und eine ILP-Formulierung ableitet, die korrekte und optimale Lösungen in Bezug auf die modellbasierte Problemdefinition garantiert (**WB 3**). Diese Konstruktionsmethodik wurde ausführlich für das statische Anwendungsszenario mit MdVNE dargestellt. Zusätzlich wurde dargelegt, dass Korrektheit und Optimalität während der kompletten Konstruktionsmethodik erhalten bleiben. Hierbei wurde zunächst ein MT-Regelsatz aus dem (UML-)Klassendiagramm erstellt, um alle Modellinstanzen generieren zu können. Danach wurde dieser Regelsatz mit EOCL-Zusicherungen verfeinert, wobei sichergestellt wird, dass diese Regeln nur dann ausgeführt werden, wenn sie keine Zusicherungen verletzen. Ausgehend von den verbliebenen OCL-Zusicherungen und der Zielfunktion wurden die Variablen und Ungleichungen für die ILP-Formulierung erstellt und die ILP-Zielfunktion abgeleitet. Diese beiden Artefakte (MT-Regelsatz und ILP-Formulierung) dienen danach als Konfigurationen für MdVNE.

Zum Schluss wurde für beide Ansätze (MdVNE und iMdVNE) ausführbarer Code erzeugt, der in einer Simulation evaluiert wurde (**WB 5**). Die Forschungs-

fragen bei dieser Evaluation zielten dabei auf den Einfluss der Modellgröße, der Menge an Anforderungen oder der Batchgröße (bei MdVNE) und der Korrektheit bzw. Optimalität der jeweiligen Implementierungen ab. Dabei zeigte sich, dass die vorgestellten modellbasierten Ansätze ähnliche Laufzeiten im Vergleich zu einer manuell und handoptimierten ILP-basierten Implementierung aufweisen. Beim statischen Anwendungsszenario benötigte MdVNE im Vergleich zur manuellen Implementierung ungefähr (konstant) doppelt soviel Zeit zum Lösen des Problems. Durch die Integration von inkrementellen Technologien bei iMdVNE konnten in einem dynamischen Szenario die Laufzeiten jedoch signifikant verringert werden, wodurch vergleichbare oder in weiten Teilen sogar geringere Laufzeiten im Vergleich zur manuellen Implementierung beobachtet wurden.

Somit wurde in dieser Arbeit erfolgreich eine modellbasierte Methodik zur Entwicklung von Algorithmen für VNE-Probleme präsentiert. Bei dieser Entwicklungsmethodik wird anhand einer Spezifikation ausführbarer Code abgeleitet, der korrekte und optimale Lösungen sicherstellt. Dieser Quellcode kann in einem Rahmenwerk ausgeführt und evaluiert werden kann. Dies verbessert die Qualität, Wartbarkeit und Anpassbarkeit der Problembeschreibung, des Quellcodes und der Lösungen. Der Entwicklungsaufwand für neue Algorithmen beruht dabei hauptsächlich auf abstrakten Spezifikationstätigkeiten und kann mit geringem Mehraufwand auch von einem statischen hin zu einem dynamischen Szenario überführt werden. Dabei konnten vergleichbare, im dynamischen Szenario sogar verringerte, Laufzeiten im Vergleich zu einem handgeschriebenen ILP-basierten Ansatz festgestellt werden.

8.2 Ausblick

Ausgehend von dieser Arbeit eröffnen sich weitere zukünftige Forschungsfelder, die wir im Folgenden diskutieren.

Rapid Prototyping von VNE-Algorithmen Aktuell existiert ein umfangreiches Rahmenwerk mit welchem sich alle möglichen Arten von Algorithmen basierend auf (i)MdVNE umsetzen lassen. Damit allerdings ausführbarer Quellcode für die (i)MdVNE-Algorithmen aus der modellbasierten Problemdefinition abgeleitet werden kann, sind zurzeit noch einige manuelle Schritte notwendig (siehe Abb. 5.1), die zukünftig weiter automatisiert werden können. So werden die MT-Regeln mit den Anwendungsbedingungen anhand der per-Konstruktion-korrekten Methodik manuell erstellt und auch die Übersetzung und Integration der COCL-Zusicherungen in die ILP-Formulierungen wird größtenteils per Hand programmiert. Die Automatisierung dieser beiden Schritte durch weitere Werkzeuge und Methoden könnte die Anwendbarkeit von (i)MdVNE noch weiter verbessern und es ermöglichen, schnell neue Algorithmen zu generieren und zu evaluieren.

An dieser Stelle werden zwei vielversprechende Ansätze zur weitgehenden Automatisierung des Entwicklungsprozesses vorgestellt. Die Integration der Software OCL2AC [74] kann den Schritt zur Ableitung von MT-Regeln ausgehend von OCL-Zusicherungen softwaretechnisch unterstützen. So können OCL-Zusi-

cherungen direkt in MT-Regeln mit integrierten Anwendungsbedingungen überführt und daraus Quellcode erzeugt werden. Einen anderen interessanten Ansatz stellt die Überführung des Metamodells und der OCL-Zusicherungen in eine Zwischensprache (z.B. Clafer) dar, um daraus direkt eine ILP-Formulierung ableiten zu können. Dieser Ansatz wird in [105] beschrieben und kann auch die Korrektheit und Optimalität während des gesamten Entwicklungsprozesses sicherstellen. Im Kontext des Sonderforschungsbereichs Multi-Mechanismen-Adaptation für das künftige Internet (MAKI)[1] werden diese Forschungsfelder weiter untersucht.

Das Anpassen und Konfigurieren der Algorithmen spielt eine entscheidende Rolle, da sich die Anforderungen zwischen unterschiedlichen Rechenzentrumsbetreibern stark unterscheiden können und auch zur Laufzeit variabel sind. Beispielsweise können sich Sicherheitsrichtlinien bei der Platzierung der virtuellen Maschinen ändern, neue Technologien (z.B. Grafikkarten, spezielle Chips für neuronale Netze oder Verschlüsselungstechnologien) integriert oder die Zielfunktion an die geschäftlichen Gegebenheiten angepasst werden. Methoden und Techniken aus dem Bereich der dynamischen (Software-)Produktlinien stellen einen vielversprechenden Ansatz zur Erstellung und Anpassung von VNE-Algorithmen dar. Somit können die einzelnen Anforderungen zueinander in Beziehung gesetzt und Analysen für konkrete Anforderungskonfigurationen durchgeführt werden. Auch die Erstellung von Quellcode oder die Ableitung von Tests kann durch Werkzeuge aus diesem Bereich unterstützt werden.

Verbesserung der Performance In dieser Arbeit wurde gezeigt, dass durch die Integration von inkrementellen Technologien die Laufzeit zur Lösung von VNE-Problemen im Gegensatz zu einer handgeschriebenen ILP-basierten Implementierung verbessert werden konnte, wobei die Eigenschaften der Korrektheit und Optimalität weiterhin sichergestellt sind. Dabei sind jedoch maßgeblich die beteiligten Werkzeuge und Technologien für die inkrementelle Ausführung und Überwachung der MT-Regeln und dem Lösen des inkrementell aktualisierten ILP-Problems verantwortlich. Im MT-Schritt könnte beispielsweise eine massive Parallelisierung bei der Ausführung und Überwachung der MT-Regel, wie es bei HiPE realisiert wird, oder der Austausch des RETE-Algorithmus durch effizientere Algorithmen wie beispielsweise TREAT weitere Performancegewinne erzielen. Da wir für die Berechnung der inkrementell aktualisierten ILP-Probleme aktuell kommerzielle ILP-Löser nutzen, ist der Einfluss auf deren Weiterentwicklungen nur begrenzt möglich. Somit wird sich in neueren Versionen dieser ILP-Löser zeigen, ob Steigerungen der Performance möglich sind.

Eine weitere Möglichkeit zur Steigerung der Effizienz liegt in der Nutzung von heuristischen Lösungsstrategien zur Einschränkung des Suchraums und (verschärften) Anforderungen, sodass die Menge an möglichen Platzierungskandidaten stark verringert werden kann. Dabei kann durch die Wahl der Heuristik bestimmt werden, ob dem ILP-Löser nach dem MT-Schritt eine Menge von annähernd optimale Lösungen oder im Extremfall nur eine mögliche Lösung übergeben wird, was erhebliche Auswirkungen auf die Laufzeit des ILP-Lösers besitzt.

[1] MAKI Webseite: `https://www.maki.tu-darmstadt.de` (besucht am: 3.9.2020)

Beispiele für verschärfte Anforderungen, die durch eine Anpassung der MT-Regeln realisiert werden können, sind die Verringerung der maximalen Pfadlänge oder dem Erlauben von Migrationen nur innerhalb desselben Racks. Da hierdurch die Menge an Möglichkeiten eingeschränkt wird, verringert sich auch die Anzahl an ILP-Variablen und -Ungleichungen und kann zu einer Performancesteigerung führen. Auch die Einschränkung auf erfolgversprechende Übereinstimmungen wie es beispielsweise durch die Integration von Fitnessfunktionen, weiteren suchbasierten Optimierungsfunktionen, genetischen Algorithmen oder dem maschinellen Lernen ermöglicht wird, kann schon bei der Ausführung der MT-Regeln die Performance drastisch erhöhen. Dabei muss für die Integration dieser Technologien lediglich das MT-Werkzeug angepasst und die Auswahl an Übereinstimmungen beispielsweise durch weitere externe Komponenten eingeschränkt werden. Durch diese Integration von heuristischen Methoden ist es möglich, mit relativ wenig Aufwand einen skalierenden heuristisch operierenden VNE-Algorithmus zu entwickeln, wobei die vorgestellten Entwicklungsansätze in weiten Teilen weiterverwendet werden können und somit die Korrektheit, jedoch nicht die Optimalität, der gefundenen Lösungen sicherstellen.

Übertragbarkeit in andere Anwendungsdomänen In dieser Arbeit wurde die Domäne von VNE-Problemen für Rechenzentren als Anwendungsbeispiel für die modellbasierte Problemdefinition der per-Konstruktion-korrekten Methodik und MdVNE bzw. iMdVNE verwendet. Jedoch ist es durch den Einsatz von universellen Modellierungssprachen, -methoden und –techniken möglich eine Vielzahl an Einbettungs-, Platzierungs-, Ressourcenallokations- oder Schedulingprobleme zu spezifizieren und zu lösen. Beispielsweise könnten diese Problemstellungen aus dem Bereich der Hardware-Software Platzierungen [35], Ressourcenmanagement in Cloudumgebungen [50], Serviceketten-Allokationsproblemen [9], Software-definierten Netzwerken [65] oder dem Prozess-Scheduling [30] stammen. Dabei müssen sich aktuell die Problemstellungen auf Graphstrukturen mit linearen Gleichungssystemen und einer linearen Zielfunktion zurückführen lassen, wobei eine regelbasierte Spezifikation von Platzierungskandidaten oder eine logikbasierte Definition von Anforderungen möglich ist und eine sinnvolle Ergänzung darstellt. Zusätzlich ist der vorgestellte Ansatz aktuell auf die Prädikatenlogik erster Stufe und einem `iterate`-Operator begrenzt, da weitere Eigenschaften, wie beispielsweise transitive Abschlüsse oder rekursive Definitionen, nicht durch die verwendete Methode aus [47] in Graph-Einschränkungen überführt werden können. Allerdings existieren beispielsweise bei [80] erste Ansätze, um transitive Abschlüsse oder rekursive Definitionen (unter bestimmten Bedingungen) mit in die Konstruktionsmethodik zu integrieren. Andere vielversprechende Ansätze finden sich beispielsweise bei [59] und [18], um Einschränkungen in Bezug auf die zu betrachtenden Pfade zu spezifizieren, umzusetzen und damit die Menge an möglichen Anwendungsdomänen zu erweitern.

LITERATURVERZEICHNIS

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 63–74, 2008. (Cited on page 13.)

[2] Edoardo Amaldi, Stefano Coniglio, Arie M. C. A. Koster, and Martin Tieves. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics*, 52:213–220, 2016. (Cited on page 2, 15, and 113.)

[3] Anthony Anjorin, Erhan Leblebici, Roland Kluge, Andy Schürr, and Perdita Stevens. A systematic approach and guidelines to developing a triple graph grammar. In *International Workshop on Bidirectional Transformations*, volume 1396 of *CEUR*, pages 81–95, 2015. (Cited on page 47.)

[4] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: advanced concepts and tools for in-place EMF model transformations. In *Intl. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, volume 6394, pages 121–135. 2010. (Cited on page 117.)

[5] Sara Ayoubi, Chadi Assi, Khaled B. Shaban, and Lata Narayanan. MINTED: multicast virtual network embedding in cloud data centers with delay constraints. *Trans. Communications*, 63(4):1291–1305, 2015. (Cited on page 5 and 29.)

[6] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Antony I. T. Rowstron. Towards predictable datacenter networks. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 242–253, 2011. (Cited on page 5 and 29.)

[7] Md. Faizul Bari, Raouf Boutaba, Rafael Pereira Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md. Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: a survey. *Communications Surveys and Tutorials*, 15(2):909–928, 2013. (Cited on page 28, 50, and 113.)

[8] Fabian Marco Bauer. Einbettung virtueller netzwerke mithilfe von inkrementellen modelltransformationstechniken, 2019. (Cited on page 98, 103, 104, and 109.)

[9] Michael Till Beck and Juan Felipe Botero. Scalable and coordinated allocation of service function chains. *Computer Communications*, 102:78–88, 2017. (Cited on page 123.)

[10] Andreas Berl, Andreas Fischer, and Hermann de Meer. Using system virtualization to create virtualized networks. *ECEASST*, 17, 2009. (Cited on page 13.)

[11] Andreas Berl, Andreas Fischer, and Hermann de Meer. Virtualisierung im future internet. *Informatik Spektrum*, 33(2):186–194, 2010. (Cited on page 14.)

[12] Jordi Cabot, Robert Clarisó, and Daniel Riera. On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*, 93:1–23, 2014. (Cited on page 7.)

[13] Wei Cai, Ryan Shea, Chun-Ying Huang, Kuan-Ta Chen, Jiangchuan Liu, Victor C. M. Leung, and Cheng-Hsin Hsu. A survey on cloud gaming: Future of computer games. *IEEE Access*, 4:7605–7620, 2016. (Cited on page 1.)

[14] Tao Chen, Xiaofeng Gao, and Guihai Chen. The features, hardware, and architectures of data center networks: A survey. *Journal of Parallel and Distributed Computing*, 96:45–74, 2016. (Cited on page 12 and 13.)

[15] II CPLEX. High-performance mathematical programming engine. *International Business Machines Corp*, 2010. (Cited on page 73 and 98.)

[16] Xiangming Dai, Ying Wang, Jason Min Wang, and Brahim Bensaou. Energy-efficient planning of qos-constrained virtual-cluster embedding in data centres. In *Intl. Conf. on Cloud Networking (CloudNet)*, pages 267–272, 2015. (Cited on page 29.)

[17] Xiangming Dai, Ying Wang, Jason Min Wang, and Brahim Bensaou. Energy efficient virtual cluster embedding in public data centers. In *Global Communications Conference (GLOBECOM)*, pages 1–6, 2015. (Cited on page 29.)

[18] Oege de Moor, David Lacey, and Eric Van Wyk. Universal regular path queries. *High. Order Symb. Comput.*, 16(1-2):15–35, 2003. (Cited on page 123.)

[19] Frederik Deckwerth and Gergely Varró. Attribute handling for generating preconditions from graph constraints. In *Intl. Conf. on Graph Transformation (ICGT)*, pages 81–96, 2014. (Cited on page 53 and 116.)

[20] Joachim Denil, Maris Jukss, Clark Verbrugge, and Hans Vangheluwe. Search-based model optimization using model transformations. In *System Analysis and Modeling: Models and Reusabilit (SAM)*, pages 80–95, 2014. (Cited on page 115.)

[21] Edsger Wybe Dijkstra. *A discipline of programming*, volume 1. Prentice Hall, 1976. (Cited on page 49 and 59.)

[22] Jun Duan and Yuanyuan Yang. Efficient virtual network embedding for variable size virtual machines in fat-tree data centers. In *Intl. Conf. on Parallel Processing (ICPP)*, pages 1–10, 2016. (Cited on page 30.)

[23] Jun Duan and Yuanyuan Yang. Placement and performance analysis of virtual multicast networks in fat-tree data center networks. *Trans. on Parallel and Distributed Systems (TPDS)*, 27(10):3013–3028, 2016. (Cited on page 29.)

[24] Jun Duan, Zhiyang Guo, and Yuanyuan Yang. Cost efficient and performance guaranteed virtual network embedding in multicast fat-tree dcns. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 136–144, 2015. (Cited on page 29.)

[25] Jun Duan, Zhiyang Guo, and Yuanyuan Yang. Embedding nonblocking multicast virtual networks in fat-tree data centers. In *Intl. Parallel and Distributed Processing Symposium (IPDPS)*, pages 123–132, 2015. (Cited on page 29.)

[26] eMoflon. HiPE: Highly (Scalable) Incremental Pattern matching Engine. `https://github.com/Arikae/HiPE-Updatesite`. (Cited on page 116.)

[27] Wenfei Fan, Xin Wang, and Yinghui Wu. Incremental graph pattern matching. *ACM Trans. Database Syst.*, 38(3):18:1–18:47, 2013. (Cited on page 116.)

[28] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. Virtual network embedding: a survey. *Communications Surveys and Tutorials*, 15(4):1888–1906, 2013. (Cited on page 2, 15, 17, 28, and 113.)

[29] Martin Fleck, Javier Troya, and Manuel Wimmer. Marrying search-based optimization and model transformation technology. *North American Symposium on Search Based Software Engineering*, 2015. (Cited on page 115.)

[30] Christodoulos A. Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: modeling, algorithms, and applications. *Ann. Oper. Res.*, 139(1):131–162, 2005. (Cited on page 123.)

[31] Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982. (Cited on page 116.)

[32] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004. (Cited on page 32 and 33.)

[33] Carlo Fuerst, Stefan Schmid, P. Lalith Suresh, and Paolo Costa. Kraken: online and elastic resource reservations for multi-tenant datacenters. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2016. (Cited on page 30 and 81.)

[34] Sebastian Götz, Johannes Mey, René Schöne, and Uwe Aßmann. A jastadd- and ilp-based solution to the software-selection and hardware-mapping-problem at the TTC 2018. In *Transformation Tool Contest (TTC)*, volume 2310 of *CEUR Workshop Proceedings*, pages 31–36. CEUR-WS.org, 2018. (Cited on page 114.)

[35] Sebastian Götz, Johannes Mey, René Schöne, and Uwe Aßmann. Quality-based software-selection and hardware-mapping as model transformation problem. In *Transformation Tool Contest (TTC)*, volume 2310 of *CEUR Workshop Proceedings*, pages 3–11. CEUR-WS.org, 2018. (Cited on page 114 and 123.)

[36] Albert G. Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 51–62, 2009. (Cited on page 13.)

[37] Albert G. Greenberg, James R. Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *Computer Communication Review*, 39(1):68–73, 2009. (Cited on page 1.)

[38] Object Management Group. *Object constraint language 2.0*. OMG, 2003. (Cited on page 7, 33, 37, and 53.)

[39] Xinjie Guan, Baek-Young Choi, and Sejun Song. Topology and migration-aware energy efficient virtual network embedding for green data centers. In *Intl. Conf. on Computer Communication and Networks (ICCCN)*, pages 1–8, 2014. (Cited on page 2, 5, 30, and 81.)

[40] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 75–86, 2008. (Cited on page 13.)

[41] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Conf. on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 15:1–15:12, 2010. (Cited on page 16, 29, and 113.)

[42] I Gurobi Optimization. Gurobi optimizer reference manual. 2016. (Cited on page 73.)

[43] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009. (Cited on page 49 and 116.)

[44] Eleni Hadjiconstantinou. Transformation of propositional calculus statements into integer and mixed integer programs: an approach towards automatic reformulation. Technical report, Brunel University, 11 1990. (Cited on page 24.)

[45] Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, Prem Prakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, Qutaibah Marwan Malluhi, Nikos Tziritas, Abhinav Vishnu, Samee U. Khan, and Albert Y.

Zomaya. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98(7):751–774, 2016. (Cited on page 2.)

[46] Eric N Hanson and Mohammed S Hasan. Gator: An optimized discrimination network for active database rule condition testing. *University of Florida.–Gainesville: CIS Departement*, 1993. (Cited on page 116.)

[47] Reiko Heckel and Annika Wagner. Ensuring consistency of conditional graph grammars-a constructive approach. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2:118–126, 1995. (Cited on page 46, 49, 53, 55, 59, 66, 67, 116, 117, and 123.)

[48] Görel Hedin. Reference attributed grammars. *Informatica (Slovenia)*, 24(3), 2000. (Cited on page 114.)

[49] IEEE. IEEE xplore digital library. URL `https://ieeexplore.ieee.org/Xplore/home.jsp`. (Cited on page 29.)

[50] Brendan Jennings and Rolf Stadler. Resource management in clouds: Survey and research challenges. *J. Network Syst. Manage.*, 23(3):567–619, 2015. (Cited on page 2 and 123.)

[51] Hiroki Kanezashi, Toyotaro Suzumura, Dario Garcia-Gasulla, Min hwan Oh, and Satoshi Matsuoka. Adaptive pattern matching with reinforcement learning for dynamic graphs. *Conf. on High Performance Computing (HiPC)*, pages 92–101, 2018. (Cited on page 116.)

[52] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Trans. Automation Science and Engineering*, 12(2):398–409, 2015. (Cited on page 1.)

[53] Timo Kehrer, Gabriele Taentzer, Michaela Rindt, and Udo Kelter. Automatically deriving the specification of model editing operations from metamodels. In *Intl. Conf. on Model Transformation (ICMT)*, pages 173–188, 2016. (Cited on page 49, 50, and 66.)

[54] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Intl. Conf. on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. (Cited on page 3 and 28.)

[55] Roland Kluge, Michael Stein, Gergely Varró, Andy Schürr, Matthias Hollick, and Max Mühlhäuser. A systematic approach to constructing incremental topology control algorithms using graph transformation. *Journal of Visual Languages and Computing (JVLC)*, 38:47–83, 2017. (Cited on page 114.)

[56] Maximilian Kratz. Evaluations-szenarien zur einbettung virtueller netzwerke in rechenzentren, 2018. (Cited on page 97.)

[57] Matthias P. Krieger and Achim D. Brucker. Extending OCL operation contracts with objective functions. *ECEASST*, 44, 2011. (Cited on page 40.)

[58] Thomas Kühne. What is a model? In *Language Engineering for Model-Driven Software Development*, 2004. (Cited on page 31 and 33.)

[59] David Lacey, Neil D. Jones, Eric Van Wyk, and Carl Christian Frederiksen. Compiler optimization correctness by temporal logic. *High. Order Symb. Comput.*, 17(3):173–206, 2004. (Cited on page 123.)

[60] Javier Larrosa and Gabriel Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical. Structures in Comp. Sci.*, 12(4):403–422, 2002. (Cited on page 115.)

[61] Federico Larumbe and Brunilde Sansò. Elastic, on-line and network aware virtual machine placement within a data center. In *Intl. Symposium on Integrated Network Management (IM)*, pages 28–36, 2017. (Cited on page 29 and 30.)

[62] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Developing emoflon with emoflon. In *Intl. Conf. on Model Transformation (ICMT)*, pages 138–145, 2014. (Cited on page 45, 62, and 72.)

[63] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Inter-model consistency checking using triple graph grammars and linear optimization techniques. In *Fundamental Approaches to Software Engineering (FASE)*, pages 191–207, 2017. (Cited on page 24, 45, 46, 62, 68, 73, 98, and 109.)

[64] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas E. Anderson. F10: A fault-tolerant engineered network. In *Symposium on Networked Systems Design and Implementation (NSDI)*, pages 399–412, 2013. (Cited on page 13.)

[65] Felipe A. Lopes, Leonidas Lima, Marcelo Santos, Robson Fidalgo, and Stenio Fernandes. High-level modeling and application validation for SDN. In *Network Operations and Management Symposium*, pages 197–205, 2016. (Cited on page 114 and 123.)

[66] Shouxi Luo, Hongfang Yu, Lemin Li, Dan Liao, and Gang Sun. Traffic-aware vdc embedding in data center: A case study of fattree. *China Communications*, 11(7):142–152, 2014. (Cited on page 5 and 30.)

[67] Bernhard Meindl and Matthias Templ. Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands*, 20, 2012. (Cited on page 98.)

[68] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 1154–1162, 2010. (Cited on page 27, 29, and 41.)

[69] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, 152:125–142, 2006. (Cited on page 76.)

[70] Daniel P. Miranker. TREAT: A better match algorithm for AI production system matching. In Kenneth D. Forbus and Howard E. Shrobe, editors, *Nat. Conf. on Artificial Intelligence*, pages 42–47. Morgan Kaufmann, 1987. (Cited on page 116.)

[71] Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998. (Cited on page 3 and 28.)

[72] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011. ISBN 978-1-118-03196-4. (Cited on page 117.)

[73] Tran Manh Nam, Nguyen Van Huynh, Le Quang Dai, and Huu-Thanh Nguyen. An energy-aware embedding algorithm for virtual data centers. In *Intl. Teletraffic Congress (ITC)*, pages 18–25, 2016. (Cited on page 5 and 30.)

[74] Nebras Nassar, Jens Kosiol, Thorsten Arendt, and Gabriele Taentzer. OCL2AC: automatic translation of OCL constraints to graph constraints and application conditions for transformation rules. In *Intl. Conf. on Graph Transformation (ICGT)*, pages 171–177, 2018. (Cited on page 117 and 121.)

[75] Leonard Nonde, Taisir EH El-Gorashi, and Jaafar MH Elmirghani. Energy efficient virtual network embedding for cloud networks. *Journal of Lightwave Technology*, 33(9):1828–1849, 2014. (Cited on page 5 and 29.)

[76] Ali Pahlevan, Xiaoyu Qu, Marina Zapater, and David Atienza. Integrating heuristic and machine-learning methods for efficient virtual machine allocation in data centers. *Trans. on CAD of Integrated Circuits and Systems*, 37(8): 1667–1680, 2018. (Cited on page 16 and 29.)

[77] Uwe Pohlmann and Marcus Hüwe. Model-driven allocation engineering (T). In *Intl. Conf. Automated Software Engineering (ASE)*, pages 374–384, 2015. (Cited on page 114.)

[78] Uwe Pohlmann and Marcus Hüwe. Model-driven allocation engineering: specifying and solving constraints based on the example of automotive systems. *Autom. Softw. Eng.*, 26(2):315–378, 2019. (Cited on page 98 and 109.)

[79] Md. Golam Rabbani, Rafael Pereira Esteves, Maxim Podlesny, Gwendal Simon, Lisandro Zambenedetti Granville, and Raouf Boutaba. On tackling virtual data center embedding problem. In *Intl. Symposium on Integrated Network Management (IM)*, pages 177–184, 2013. (Cited on page 16, 29, and 113.)

[80] Hendrik Radke, Thorsten Arendt, Jan Steffen Becker, Annegret Habel, and Gabriele Taentzer. Translating essential OCL invariants to nested graph constraints for generating instances of meta-models. *Science of Computer Programming*, 152:38–62, 2018. (Cited on page 37, 49, 53, 55, 56, 59, 66, 67, 117, and 123.)

[81] T. K. R. K. Rao, S. A. Khan, Z. Begum, and C. Divakar. Mining the e-commerce cloud: a survey on emerging relationship between web mining, e-commerce and cloud computing. In *Intl. Conference on Computational Intelligence and Computing Research*, pages 1–4, 2013. (Cited on page 1.)

[82] Arend Rensink, Ákos Schmidt, and Dániel Varró. Model checking graph transformations: a comparison of two approaches. In *Intl. Conf. on Graph Transformation (ICGT)*, volume 3256, pages 226–241. 2004. (Cited on page 117.)

[83] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *Intl. Joint Conference on INC, IMS and IDC*, pages 44–51, 2009. (Cited on page 1.)

[84] Matthias Rost, Carlo Fuerst, and Stefan Schmid. Beyond the stars: revisiting virtual cluster embeddings. *Computer Communication Review*, 45(3):12–18, 2015. (Cited on page 29.)

[85] Sahel Sahhaf, Wouter Tavernier, Matthias Rost, Stefan Schmid, Didier Colle, Mario Pickavet, and Piet Demeester. Network service chaining with optimized network function embedding supporting service decompositions. *Computer Networks*, 93:492–505, 2015. (Cited on page 20 and 97.)

[86] Thomas Schnabel, Markus Weckesser, Roland Kluge, Malte Lochau, and Andy Schürr. Cardygan: tool support for cardinality-based feature models. In *Intl. Workshop on Variability Modelling of Software-intensive Systems*, pages 33–40. ACM, 2016. (Cited on page 73.)

[87] René Schöne, Sebastian Götz, Uwe Aßmann, and Christoff Bürger. Incremental runtime-generation of optimisation problems using rag-controlled rewriting. In *Intl. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, volume 1742 of *CEUR Workshop Proceedings*, pages 26–34. CEUR-WS.org, 2016. (Cited on page 114.)

[88] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998. (Cited on page 20.)

[89] Andy Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science*, pages 151–163, 1994. (Cited on page 45, 46, and 62.)

[90] Arman Shehabi, Sarah Josephine Smith, Dale A. Sartor, Richard E. Brown, Magnus Herrlin, Jonathan G. Koomey, Eric R. Masanet, Nathaniel Horner, Inês Lima Azevedo, and William Lintner. United states data center energy usage report. Technical report, 06/2016 2016. (Cited on page 1.)

[91] Siqi Shen, Vincent van Beek, and Alexandru Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *Intl. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 465–474, 2015. (Cited on page 86 and 97.)

[92] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development - technology, engineering, management*. Pitman, 2006. (Cited on page 31, 32, and 33.)

[93] Daniel Strüber. Generating efficient mutation operators for search-based model-driven engineering. In *Intl. Conf. on Model Transformation (ICMT)*, pages 121–137, 2017. (Cited on page 115.)

[94] Yantao Sun, Jing Cheng, Qiang Liu, and Weiwei Fang. Diamond: An improved fat-tree architecture for large-scale data centers. *Journal of Communications*, 9(1):91–98, 2014. (Cited on page 13.)

[95] Andrew S Tanenbaum and David J Wetherall. *Computernetzwerke*. München: Pearson, 2012. (Cited on page 11.)

[96] Stefan Tomaszek, Roland Speith, and Andy Schürr. Virtual network embedding: ensuring correctness and optimality by construction using model transformation and integer linear programming techniques. *Software and Systems Modeling [accepted for publication]*. (Cited on page 20, 31, and 43.)

[97] Stefan Tomaszek, Erhan Leblebici, Lin Wang, and Andy Schürr. Model-driven development of virtual network embedding algorithms with model transformation and linear optimization techniques. In *Modellierung 2018*, pages 39–54, 2018. (Cited on page 43.)

[98] Stefan Tomaszek, Erhan Leblebici, Lin Wang, and Andy Schürr. Virtual network embedding: reducing the search space by model transformation techniques. In *Intl. Conf. on Model Transformation (ICMT)*, pages 59–75, 2018. (Cited on page 43.)

[99] Stefan Tomaszek, Lars Fritsche, and Andy Schürr. Dynamic virtual network embedding: using incremental model transformation and integer linear programming techniques. *Journal of Object Technology*, 19(2), 2020. (Cited on page 71.)

[100] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Software and System Modeling*, 15(3):609–629, 2016. (Cited on page 72 and 116.)

[101] Gergely Varró and Frederik Deckwerth. A rete network construction algorithm for incremental pattern matching. In *Intl. Conf. on Model Transformation (ICMT)*, pages 125–140, 2013. (Cited on page 116.)

[102] Gergely Varró, Dániel Varró, and Andy Schürr. Incremental graph pattern matching: Data structures and initial experiments. *Electronic Communications of the EASST*, 4, 2006. (Cited on page 116.)

[103] Yating Wang, Ing-Ray Chen, and Ding-Chau Wang. A survey of mobile cloud computing applications: Perspectives and challenges. *Wireless Personal Communications*, 80(4):1607–1623, 2015. (Cited on page 1.)

[104] Jos B. Warmer and Anneke G. Kleppe. *Object constraint language 2.0*. Software-Entwicklung. mitp-Verl., 2004. ISBN 3-8266-1445-3 and 978-3-8266-1445-3. (Cited on page 37.)

[105] Markus Weckesser. *Automatisierte Analyse integrierter Software-Produktlinien-Spezifikationen*. PhD thesis, Darmstadt University of Technology, Germany, 2019. (Cited on page 98, 109, and 122.)

[106] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998. ISBN 9780471283669. (Cited on page 20.)

[107] Di Xie, Ning Ding, Y. Charlie Hu, and Ramana Rao Kompella. The only constant is change: incorporating time-varying network reservations in data centers. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 199–210, 2012. (Cited on page 30.)

[108] Jing Xu and José A. B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Intl. Conf. on Green Computing and Communications (GreenCom)*, pages 179–188, 2010. (Cited on page 3, 28, and 29.)

[109] Yang Yang, Xiaolin Chang, Jiqiang Liu, and Lin Li. Towards robust green virtual cloud data center provisioning. *Trans. on Cloud Computing*, 5(2): 168–181, 2017. (Cited on page 5, 16, 30, and 86.)

[110] Z. Yang and Y. Guo. An exact virtual network embedding algorithm based on integer linear programming for virtual network request with location constraint. *China Communications*, 13(8):177–183, 2016. (Cited on page 3 and 28.)

[111] Hanene Ben Yedder, Qingye Ding, Umme Zakia, Zhida Li, Soroush Haeri, and Ljiljana Trajkovic. Comparison of virtualization algorithms and topologies for data center networks. In *Intl. Conf. on Computer Communication and Networks (ICCCN)*, pages 1–6, 2017. (Cited on page 28.)

[112] Lei Yu and Zhipeng Cai. Dynamic scaling of virtual clusters with bandwidth guarantee in cloud datacenters. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2016. (Cited on page 5 and 30.)

[113] Lei Yu and Haiying Shen. Bandwidth guarantee under demand uncertainty in multi-tenant clouds. In *Intl. Conf. on Distributed Computing Systems (ICDCS)*, pages 258–267, 2014. (Cited on page 29.)

[114] Ruozhou Yu, Guoliang Xue, Xiang Zhang, and Dan Li. Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2017. (Cited on page 29.)

[115] Ying Yuan, Cong Wang, Sancheng Peng, and Keshav Sood. Topology-oriented virtual network embedding approach for data centers. *IEEE Access*, 7: 2429–2438, 2019. (Cited on page 30.)

[116] Pamela Zave. Understanding SIP through Model-Checking. In *Principles, systems and applications of IP telecommunications. Services and security for next generation networks*, volume 5310, pages 256–279. 2008. (Cited on page 117.)

[117] Pamela Zave. Using lightweight modeling to understand chord. *SIGCOMM Computer Communication Review*, 42(2):49–57, 2012. (Cited on page 117.)

[118] Deze Zeng, Song Guo, Huawei Huang, Shui Yu, and Victor C.M. Leung. Optimal vm placement in data centers with architectural and resource constraints. *Intl. Journal of Autonomous and Adaptive Communications Systems (IJAACS)*, 8(4):392–406, 2015. (Cited on page 2, 5, 16, 17, 20, 27, 29, 98, 109, and 113.)

[119] Bolei Zhang, Zhuzhong Qian, Wei Huang, Xin Li, and Sanglu Lu. Minimizing communication traffic in data centers with power-aware VM placement. In *Intl. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 280–285, 2012. (Cited on page 5 and 29.)

[120] Qi Zhang, Mohamed Faten Zhani, Maissa Jabri, and Raouf Boutaba. Venice: reliable virtual data center embedding in clouds. In *Intl. Conf. on Computer Communications (INFOCOM)*, pages 289–297, 2014. (Cited on page 16 and 29.)

[121] Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *Int. J. Communication Systems*, 26(8):1054–1073, 2013. (Cited on page 3 and 28.)

[122] Mohamed Faten Zhani, Qi Zhang, Gwendal Simon, and Raouf Boutaba. VDC planner: dynamic migration-aware virtual data center embedding for clouds. In *Intl. Symposium on Integrated Network Management (IM)*, pages 18–25, 2013. (Cited on page 30.)

[123] Biyu Zhou, Jie Wu, Fa Zhang, and Zhiyong Liu. Resource optimization for survivable embedding of virtual clusters in cloud data centers. In *Intl. Conf. on Performance Computing and Communications (IPCCC)*, pages 1–8, 2017. (Cited on page 2 and 29.)

[124] Jing Zhu, Dan Li, Jianping Wu, Hongnan Liu, Ying Zhang, and Jingcheng Zhang. Towards bandwidth guarantee in multi-tenancy cloud computing networks. In *Intl. Conf. on Network Protocols (ICNP)*, pages 1–10, 2012. (Cited on page 30.)

[125] Steffen Zschaler and Lawrence Mandow. Towards model-based optimisation: using domain knowledge explicitly. In *Software Technologies: Applications and Foundations (STAF), Collocated Workshops*, pages 317–329, 2016. (Cited on page 115.)

[126] Albert Zündorf. Graph pattern matching in progres. In *Workshop on Graph Gramars and Their Application to Computer Science*, pages 454–468, 1996. ISBN 3-540-61228-9. (Cited on page 115.)

[127] Cheng Zuo, Hongfang Yu, and Vishal Anand. Reliability-aware virtual data center embedding. In *Intl. Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 151–157, 2014. (Cited on page 29.)